Un enfoque MDD para el desarrollo de aplicaciones móviles nativas enfocadas en la capa de datos

Proyecto Final de Carrera

Lic. Manuel Núñez

Departamento de Electrónica e Informática (DEI)
Universidad Católica "Nuestra Señora de la Asunción"

Esta tesis se presenta para el grado de
Ingeniería en Informática

Facultad de Ciencias y Tecnología
(CyT) Mayo 2017

Resumen

Diariamente nos beneficiamos con el uso de los teléfonos móviles inteligentes, constituyendo una parte esencial de nuestra vida diaria. La variedad de teléfonos móviles inteligentes disponibles hoy día en el mercado es muy amplia. Este ambiente fragmentado es uno de los principales retos para los desarrolladores de aplicaciones móviles en la actualidad, donde el esfuerzo de desarrollo de aplicaciones específicas para cada plataforma se incrementa casi linealmente ante tan profundas diferencias, afectando la portabilidad de las mismas y teniendo que desarrollar cada aplicación prácticamente de cero. Por otra parte, el desarrollo de aplicaciones móviles implica manejar los recursos limitados del dispositivo, en especial, la conectividad. La mayoría de las aplicaciones se encuentran en constante conexión con datos remotos, y no tener en cuenta este aspecto puede resultar en aplicaciones con poca usabilidad, que sin conexión de red quedan prácticamente inutilizables, sin contenido para mostrar. Ante esto, se recomienda usar el almacenamiento en caché para mejorar el rendimiento de la aplicación, permitiendo el acceso a la información y servicios cuando no haya conexión a red.

Del estudio que llevamos a cabo, destacamos la necesidad de especificaciones para describir la persistencia en el modelado de aplicaciones móviles y la poca adopción de la Arquitectura Dirigida por Modelos (MDA). Pretendemos cubrir conceptualmente la persistencia en el desarrollo de aplicaciones móviles, mediante elementos específicos que permitan tener en cuenta las distintas opciones y mecanismos de persistencia en el modelado, a fin de posibilitar el almacenamiento de datos incluso sin conexión de red, y considerando la conexión con fuentes remotas. En una arquitectura multicapas, estos conceptos se ubicarían en la capa de datos.

Existen distintas tendencias para el desarrollo de aplicaciones móviles. Destacando aquellos enfoques (semi)automatizados, encontramos los *frameworks* y los enfoques de Desarrollo Dirigido por Modelos (MDD). Como parte del marco investigativo de este proyecto, el Enfoque de Desarrollo Web Orientado a Modelos (MoWebA) es un enfoque metodológico MDD para el desarrollo de aplicaciones web que adopta el estándar MDA. Entre los aspectos de MoWebA que podrían tener impacto positivo en el problema de la portabilidad en el desarrollo de aplicaciones móviles, destacamos el enriquecimiento de los

modelos existentes en orden a considerar aspectos relacionados a la arquitectura final del sistema, gracias a su Modelo Específico de la Arquitectura (ASM). Mediante este modelo se podría analizar la posibilidad de representar aplicaciones móviles, utilizando conceptos específicos de la arquitectura y plataforma móvil.

Este Proyecto Final de Carrera propone un enfoque MDD, mediante la extensión de MoWebA, para el desarrollo de aplicaciones móviles nativas enfocadas en la capa de datos. Nuestra propuesta MoWebA Mobile parte de la extensión del metamodelo y perfil de MoWebA. A partir de las extensiones al Modelo Independiente de la Plataforma (PIM) de MoWebA, obtuvimos el ASM móvil para el desarrollo de aplicaciones móviles enfocadas en la capa de datos, posibilitando el diseño de la persistencia así como los proveedores de datos de la aplicación móvil.

Palabras claves: Desarrollo Dirigido por Modelos, Desarrollo Móvil Dirigido por Modelos, Aplicaciones Móviles, Persistencia de Datos, Proveedores de Datos, Capa de Datos.

Tabla de Contenido

Lista de figuras

Lista de cuadros

Lista de acrónimos

IDE Integrated Development Environment.

MDD Model Driven Development.

OMG Object Management Group.

MDA Model Driven Architecture.

DSM Domain Specific Modeling.

CIM Computation Independent Model.

MDSE Model Driven Software Engineering.

ASM Architecture Specific Model.

PIM Platform Independent Model.

PSM Platform Specific Model.

ISM Implementation Specific Model.

API Application Programming Interface.

SDK Software Developer Kit.

IDC International Data Corporation.

UX User Xperience.

DSL Domain Specific Language.

GUI Graphic User Interface.

MoWebA Model Oriented Web Approach.

Capítulo 1

Introducción

Diariamente nos beneficiamos con el uso de los teléfonos móviles inteligentes[1], o *smartphones* en inglés. Estos teléfonos móviles con avanzados recursos computacionales son considerados como una parte esencial de nuestra vida diaria, brindándonos ayuda tanto en el ámbito personal, laboral como educacional. Las personas utilizan sus teléfonos ya no simplemente para llamar o enviar mensajes de texto, sino para navegar en Internet, usar redes sociales, jugar, escuchar músicas, ver y compartir fotos y vídeos, acceder a sus correos, realizar transacciones bancarias, geolocalización, o interactuar con diferentes tipos de sistemas complejos [5].

El auge de las aplicaciones móviles y la facilidad con la que hoy en día uno puede acceder a un teléfono móvil inteligente, promueve a que año tras año vaya creciendo el número de usuarios de éstos dispositivos; y de la misma forma, se produce el aumento del volumen de ventas en el mercado incentivados por la demanda y por la economía que mueve este sector tecnológico. Sin duda, este sector es de gran importancia y no está libre de problemas, sino en constante cambio y evolución [15].

El mercado móvil y el auge por los teléfonos móviles inteligentes y aplicaciones, tienen un gran impacto en nuestra sociedad tecnológica. Enfocar nuestro estudio aquí es de fundamental importancia, por su relevancia en la actualidad, en especial al sector informático.

Las siguientes secciones se distribuyen en: en la sección 1.1 se presenta el contexto del proyecto en el cual se enmarca nuestro Proyecto Final de Carrera o PFC. En la sección 1.2 se describe los problemas considerados en el desarrollo de aplicaciones móviles. En la sección 1.3 se expone la solución a los problemas planteados, estableciendo el objetivo general del trabajo. Por último, presentamos la estructura de capítulos del resto de este documento.

[1] Teléfono móvil con avanzados recursos computacionales y equipado con tecnologías que facilitan el acceso a internet, corren aplicaciones, son táctiles, poseen cámara y otros sensores, todos bajo un avanzado sistema operativo [48].

1.1 Contexto

Este trabajo se enmarca dentro del proyecto MDD+ "Mejorando el Proceso de Desarrollo de Software: Propuesta basada en MDD", nro. de referencia 14-INV-056. Este PFC es financiado por el Consejo Nacional de Ciencia y Tecnología (CONACYT, Paraguay) a través del programa PROCIENCIA con recursos del Fondo para la Excelencia de la Educación e Investigación (FEEI) del FONACIDE, bajo la supervisión del investigador principal Dr.Luca Cernuzzi.

MDD+[2] busca explorar los beneficios de la aplicación MDD en el desarrollo de aplicaciones software de buena calidad, utilizando tecnologías actuales. Abarca una serie de proyectos dentro del área común de investigación, MDD. Este PFC se enfoca en el desarrollo de aplicaciones móviles y MDD. De la misma forma, existen otros proyectos en el área: RIAs, Mobile Cloud computing, Testing y M2M Transformations.

Marco investigativo del proyecto: MoWebA

MoWebA sirvió como punto de partida para el desarrollo de este trabajo: extendimos el enfoque de desarrollo web de MoWebA al ambiente móvil. Iniciamos el desarrollo de la propuesta MoWebA Mobile para el desarrollo de aplicaciones móviles, actualmente solo enfocándonos en la capa de datos. Más descripción de esta propuesta puede ser encontrada en el capítulo 5: "Una Extensión de MoWebA para el Desarrollo de Aplicaciones Móviles".

1.2 Problemáticas

La variedad de teléfonos móviles inteligentes disponibles hoy día en el mercado es muy amplia. El número de sistemas operativos móviles no es tan elevado, pero las diferentes versiones de un mismo sistema operativo incrementan la variabilidad con la que se debe lidiar al momento de desarrollar aplicaciones. Cada sistema operativo móvil presenta un Entorno de Desarrollo Integrado, del inglés *Integrated Development Environment* (IDE) propio, con un diseño de interfaz diferente; así también, el manejo de los recursos de hardware y de los datos en las aplicaciones suele realizarse de manera diferente en cada sistema [21]. Este ambiente fragmentado es uno de los principales retos para los desarrolladores de aplicaciones móviles en la actualidad, donde la variedad de plataformas[3] origina este fenómeno conocido como **fragmentación** [59] [40] [51] [49]. Encontramos así que el esfuerzo de desarrollo de aplicaciones específicas para cada plataforma (desarrollo de aplicaciones nativas) se

[2]Proyecto MDD+. http://www.dei.uc.edu.py/proyectos/mddplus/

[3]Una plataforma móvil comprende el hardware subyacente del dispositivo, la arquitectura sobre la que está basada, el sistema operativo, el Kit de Desarrollo de Software, del inglés *Software Developer Kit* (SDK) del proveedor y las librerías estándares [55].

incrementa casi linealmente ante tan profundas diferencias, afectando la portabilidad de las mismas y teniendo que desarrollar cada aplicación prácticamente de cero, lo que implica un incremento en el tiempo de desarrollo y en el costo de mantenimiento de las aplicaciones [24] [40].

Lo que provoca que el desarrollo de aplicaciones móviles sea diferente son ciertos requerimientos adicionales, que son más acentuados en este tipo de aplicaciones en comparación con otras aplicaciones de software tradicionales (por ejemplo, aplicaciones de escritorio): manejan sensores (por ejemplo, giróscopo, brújula, entre otros), trabajan con recursos limitados (como la memoria del teléfono, la batería y la conectividad, entre otros), son dirigidas por eventos e interactúan con otras aplicaciones [5][8].

Los recursos limitados del teléfono móvil son aspectos importantes a tener en cuenta al momento de desarrollar una aplicación móvil. En particular, destacamos la conectividad. La mayoría de las aplicaciones se encuentran en constante conexión con datos remotos, y en estos, la conectividad de red juega un papel crucial: no tener en cuenta estos aspectos puede resultar en aplicaciones con poca usabilidad, que ante sin conexión de red quedan prácticamente inutilizables, sin contenido para mostrar. *Microsoft Application Architecture Guide* o MAAG [46], recomienda usar el almacenamiento en caché para mejorar el rendimiento y *responsiveness*[4] de la aplicación, y para permitir operaciones cuando no haya conexión a red.

El desarrollo de aplicaciones móviles generalmente involucra guardar datos. Estos datos pueden ser de distintos tipos: desde imágenes provenientes de la cámara, a configuraciones de preferencia del usuario, respaldo de archivos, documentos, entre otros. Estos datos pueden provenir de una fuente externa (por ejemplo, un servidor remoto), los cuales pueden ser manipulados localmente; o bien, pueden provenir del mismo teléfono (por ejemplo, los sensores) [40]. Actualmente, son varias las opciones de almacenamiento que podemos encontrar: i) HTML5[5] y sus métodos específicos de persistencia dentro de las aplicaciones web; ii) almacenamiento local con archivos y bases de datos integradas como SQLite[6]; y, iii) almacenamiento externo con servicios en la nube (como Dropbox[7] y Google Drive[8]), y dispositivos de almacenamiento externo (por ejemplo, tarjetas de memoria SD) [17]. A su vez, cada sistema operativo móvil maneja estas opciones de persistencia siguiendo diferentes caminos, dificultando el proceso de desarrollo en este ambiente fragmentado [21] [38].

[4] *Responsiveness*, o la expectativa que se tiene de las aplicaciones móviles de que provean acceso a la información y servicios en cualquier momento y lugar [63][27].

[5] HTML5. http://www.w3.org/TR/html5/

[6] SQLite. https://sqlite.org/

[7] Dropbox. https://www.dropbox.com

[8] Google Drive. https://drive.google.com

1.3 Solución propuesta

El Desarrollo Dirigido por Modelos, del inglés *Model Driven Development* (MDD) permite ir un paso atrás y re-compilar el modelo conceptual a diferentes plataformas tecnológicas usando diferentes Modelo Específico de la Plataforma, del inglés *Platform Specific Model* (PSM)s y compiladores, asegurando portabilidad [43]. La adopción de MDD corresponde a una herramienta de solución para minimizar la brecha de heterogeneidad que supone la fragmentación, y de esa forma reducir el esfuerzo de desarrollo de las aplicaciones móviles nativas entre plataformas. La idea que nos presenta MDD es describir un problema en un modelo y generar software a partir de esta representación [56] [47] [11]. Desarrollar aplicaciones que comparten las mismas funcionalidades y comportamiento, pero para diferentes plataformas, constituye un área adecuado para esta metodología [5] [49]. MDD trata el problema de la redundancia de tareas, reduciendo el esfuerzo de programación y los errores de codificación [18]. Así, el uso de metodologías basadas en MDD, frente las problemáticas presentadas, puede suponer facilidades en el desarrollo de aplicaciones para teléfonos inteligentes [49].

Atendiendo a lo dicho previamente, el Enfoque de Desarrollo Web Orientado a Modelos, del inglés *Model Oriented Web Approach* (MoWebA) es un enfoque metodológico MDD para el desarrollo de aplicaciones web que adopta la Arquitectura Dirigida por Modelos, del inglés *Model Driven Architecture* (MDA) [20]. Sanchiz et al. [53] expresan que esta metodología podría constituirse en una opción adecuada frente al problema de la portabilidad en el desarrollo de aplicaciones móviles, gracias a su estructura por capas bien definida (mediante la cual se logra una clara separación de conceptos), el modelado centrado en la navegación jerárquica orientada a funciones (permite un diseño más adecuado de una navegación basada en la interacción del usuario o del contexto) y el enriquecimiento de los modelos existentes en orden a considerar aspectos relacionados a la arquitectura final del sistema (por ejemplo, Rich Internet Applications (RIAs), Service Oriented Applications (SOA), REST, entre otros), gracias a su Modelo Específico de la Arquitectura, del inglés *Architecture Specific Model* (ASM) [20]. Mediante este modelo se podría analizar la posibilidad de representar aplicaciones móviles, utilizando conceptos específicos de la arquitectura móvil.

Siguiendo la arquitectura multicapas de MAAG [46], enmarcamos el estudio de la persistencia dentro de la capa de datos. Esta capa, a parte del manejo de la persistencia local de la aplicación, define distintos proveedores de datos para la aplicación móvil.

1.4 Objetivos

A partir de lo mencionado, en este PFC nos planteamos los siguientes objetivos:

Objetivo General

- Definir un enfoque MDD, mediante la extensión de MoWebA, para el desarrollo de aplicaciones móviles nativas enfocadas en la capa de datos.

Objetivos Específicos

- Analizar las propuestas MDD para el desarrollo de aplicaciones móviles, observando los aspectos de persistencia utilizados.
- Extender MoWebA para el desarrollo de aplicaciones móviles, considerando la capa de datos mediante metamodelos y perfiles ASM, y el establecimiento de reglas de transformación para tal propósito.
- Realizar un análisis de la propuesta mediante la valoración de resultados aplicados a un caso de prueba.

1.5 Estructura del trabajo

Este documento se encuentra organizado de la siguiente manera:

- El capítulo 2 introduce el desarrollo de las aplicaciones móviles, expone los problemas asociados al proceso de desarrollo, y describe las tendencias actuales para el desarrollo de aplicaciones móviles.
- El capítulo 3 analiza la persistencia de datos en los teléfonos móviles inteligentes, analiza el problema asociado, y los distintos mecanismos de persistencia existentes.
- El capítulo 4 presenta el enfoque MDD para el desarrollo de aplicaciones móviles, analiza propuestas MDD para el desarrollo de aplicaciones móviles y discute algunos aspectos resaltantes encontrados; y presenta a MoWebA para el desarrollo de aplicaciones móviles.
- El capítulo 5 presenta a MoWebA Mobile, nuestro enfoque para el desarrollo de aplicaciones móviles nativas enfocadas en la capa de datos; describe los aspectos considerados y el proceso de desarrollo a seguir con el enfoque; detalla las extensiones

realizadas a MoWebA y presenta el ASM móvil; presenta el modelado de una aplicación a partir de nuestro enfoque; describe las reglas de transformación y el código generado a partir de los modelos definidos.

- El capítulo 6 presenta la validación preliminar de MoWebA Mobile, con el propósito de realizar un primer análisis de la propuesta y así obtener un primer juicio de intuiciones en cuanto a su usabilidad y portabilidad; y describe los resultados obtenidos.
- El capítulo 7 presenta la conclusión de este PFC y los posibles trabajos futuros a realizar a partir de este proyecto.

Capítulo 2

El desarrollo de aplicaciones móviles

En este capítulo se presenta una visión general de los conceptos necesarios para adentrarnos al mundo del desarrollo de las aplicaciones móviles. Además, se exponen los problemas asociados al proceso de desarrollo, y las distintas opciones disponibles para encarar el desarrollo de aplicaciones móviles.

En la sección 2.1 se presenta a la aplicación móvil y los conceptos relacionados al mismo. Además, se comentan los sistemas operativos móviles actualmente disponibles en el mercado; y por último, se dan detalles de la arquitectura típica, que por lo general, poseen las aplicaciones móviles.

Continuando el proceso de desarrollo de una aplicación móvil, en la sección 2.2 se presentan los distintos enfoques de desarrollo disponibles a adoptar al momento de encarar el desarrollo de estas aplicaciones. Cada una presenta ventajas y desventajas, y por lo general, es un punto muy relevante desde donde parte y se gesta todo el proceso de desarrollo.

En la sección 2.3 se describe uno de los principales problemas que afectan el proceso de desarrollo de las aplicaciones móviles: la fragmentación. Se describen las causas y se habla del caso específico de Android y su problema con la fragmentación.

Por último, en la sección 2.4 se presenta qué alternativas de desarrollo encontramos actualmente disponibles y una comparación entre ellos, brindando detalles sobre los aspectos más destacados encontrados.

2.1 Las aplicaciones móviles

Los teléfonos móviles inteligentes son teléfonos móviles con avanzados recursos computacionales y equipados con tecnologías que facilitan el acceso a Internet, corren aplicaciones, son táctiles, poseen cámara y otros sensores, todos bajo un avanzado sistema operativo [48]. Cuenta con un número de aplicaciones preinstaladas, propias del sistema

operativo; de igual forma, aplicaciones de terceros pueden ser igualmente descargadas e instaladas desde tiendas virtuales [61]. Las tiendas virtuales, especialmente diseñadas por las compañías desarrolladoras de los sistemas operativos, permiten el acceso a contenidos para sus usuarios, así como para la comercialización de las mismas. Como principales tiendas virtuales tenemos el Google Play Store[1], exclusivo para usuarios de Android; y el App Store[2], exclusivo para usuarios de iOS.

Lo que provoca que el desarrollo de aplicaciones móviles sea diferente, son ciertos requerimientos adicionales que son más acentuados en este tipo de aplicaciones en comparación con otras aplicaciones de software tradicionales (por ejemplo, aplicaciones de escritorio): manejan sensores (por ejemplo, giróscopo, GPS, movimiento, luz, aceleración, entre otros) y hardware específico del teléfono inteligente (por ejemplo, cámara, batería, Bluetooth, NFC[3], entre otros); trabajan con recursos limitados (como la memoria del teléfono, la batería y la conectividad, entre otros); son dirigidas por eventos e interactúan con otras aplicaciones [5][8].

Los recursos limitados del teléfono móvil son aspectos importantes a tener en cuenta al momento de desarrollar una aplicación móvil. En particular, destacamos la conectividad. Generalmente, los teléfonos inteligentes requieren de una conexión activa a Internet para realizar peticiones a servicios web y procesar las respuestas. En caso que la red no esté disponible, es necesario contar con almacenamiento local (en modo *offline* o "sin conexión"). En el capítulo 3 hablaremos del problema asociado al manejo de este recurso y las soluciones al mismo.

2.1.1 Sistemas operativos móviles

Actualmente los sistemas operativos más utilizados en los teléfonos móviles inteligentes son iOS (desarrollado por Apple), Android (de Google) y Windows Phone (de Microsoft). Cada uno de estos sistemas poseen sus propios lenguajes de programación, IDEs, SDKs[4] e Interfaz de Programación de Aplicaciones, del inglés *Application Programming Interface* (API)s[5]. Así sabemos que desarrollar aplicaciones para iOS requiere de Swift, Android de Java y Windows Phone de .NET. Además de los ya mencionados, existen otros sistemas operativos

[1] Google Play Store. http://play.google.com

[2] App Store. https://itunes.apple.com/es/genre/ios/id36?mt=8

[3] *Near Field Communication*. https://developer.android.com/guide/topics/connectivity/nfc/index.html

[4] El SDK es un conjunto de herramientas que permiten la creación de aplicaciones para cierto sistema operativo, paquete de software o *framework*. Fuente: https://en.wikipedia.org/wiki/Software_development_kit

[5] La API consiste en una capa de abstracción por el que se provee un conjunto de funciones, procedimientos y protocolos para el acceso a ciertas bibliotecas para que puedan ser accedidos por otro software. Fuente: https://en.wikipedia.org/wiki/Application_programming_interface

presentes en el mercado móvil actual, como FireOS (de Amazon), Blackberry OS (de Blackberry), Tizen (de Samsung) y FirefoxOS (de Mozilla) [24].

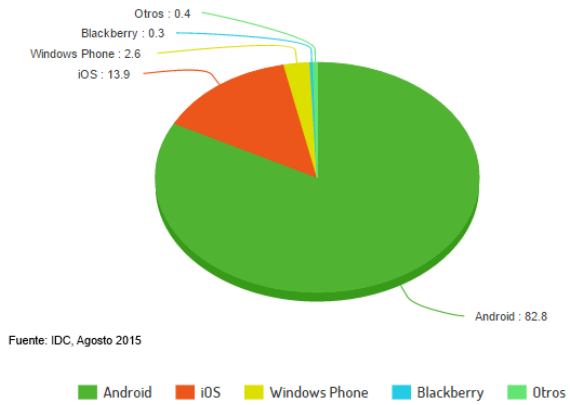


Figura 2.1 Mercado actual de los sistemas operativos móviles

Como se puede observar en la figura 2.1, según estudios realizados por la *International Data Corporation* (IDC), en su informe del mercado de los sistemas operativos de teléfonos móviles inteligentes[6], la cuota de mercado es bien diferenciada para cada sistema operativo, siendo Android quien domina el mercado con un 82.8%, seguido por iOS con un 13.9%, Windows Phone con un 2.6% y Blackberry OS con un 0.3%. Otros constituirían el 0.4% restante.

2.1.2 Arquitectura de las aplicaciones

Acorde a MAAG [46], una aplicación móvil normalmente estará estructurada como una aplicación multicapas en: capa de presentación, capa de negocios y capa de datos. La figura 2.2 nos muestra la estructura o arquitectura típica de una aplicación móvil. Por lo general, la aplicación móvil contiene componentes de interfaces de usuario (o *UI Componentes*) en la capa de presentación, y quizás pueda incluir componentes lógicos de presentación (o *Presentation Logic Components*). La capa de negocios, si existe, contendrá componentes de lógica de negocio (o *Business Components*), el flujo de negocios (o *Business Workflow*), entidades de negocio (o *Business Entities*) que son requeridas por la aplicación y la estética de la aplicación (o *Application Façade*). Por otra parte, la capa de datos usualmente incluye componentes de acceso a datos (o *Data Access Components*) y agentes de servicio (o *Service Agents*). En el capítulo 3.3 brindaremos más detalles acerca de la capa de datos, cómo funciona y los componentes involucrados.

Una aplicación móvil puede o no contemplar una división arquitectural como la propuesta por MAAG, inclusive ciertos aspectos podrían no ser tenidos en cuenta en el diseño. Como

[6]Informe del mercado móvil de la IDC. http://ow.ly/PPta307vlRf

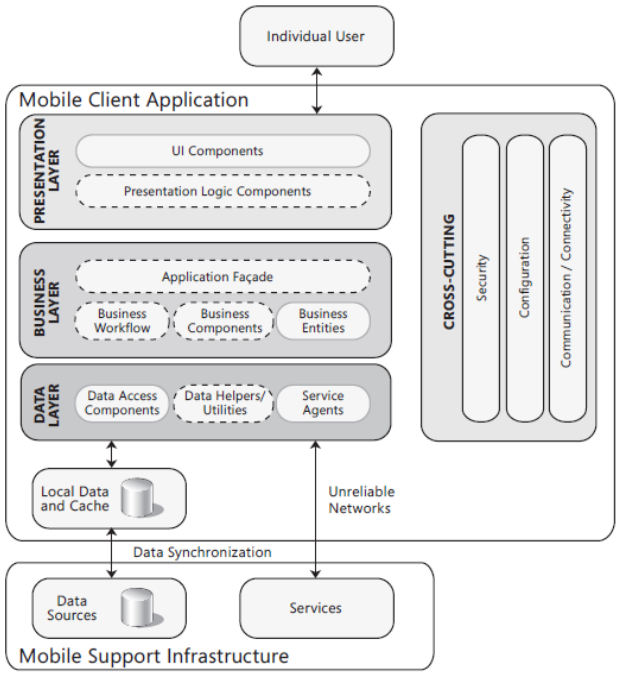


Figura 2.2 Estructura típica de una aplicación móvil. *Fuente: Microsoft Application Architecture Guide [46]*

veremos en la sección 2.4, existen herramientas y propuestas para el diseño de aplicaciones móviles, pero no todas estas contemplan el diseño de todas las capas.

2.2 Enfoques de desarrollo

Existen distintos enfoques de desarrollo de aplicaciones móviles. Se puede desarrollar aplicaciones nativas para cada posible plataforma, o desarrollar una vez una aplicación web para todas las plataformas, o bien, desarrollar aplicaciones con lo mejor de cada uno de estos enfoques mencionados. Cada uno de estos enfoques presenta ventajas y desventajas, y su elección depende de factores como el tipo de aplicación a generar, el tiempo de desarrollo, el esfuerzo, entre otros. A continuación hablaremos de cada uno de estos enfoques.

2.2.1 Desarrollo de aplicaciones nativas

Desarrollar una aplicación nativa para distintas plataformas se ha dificultado para las compañías. Con la variedad de sistemas operativos, se necesita desarrollar la misma aplicación para cada una de ellos. Puesto que no se puede reutilizar el mismo código fuente para otras plataformas, la misma aplicación debe ser generada desde cero, utilizando

lenguajes y SDK específicos, provistos por algunos IDEs como Eclipse, Android Studio[7], Xcode[8] y Visual Studio[9]. Esto demanda más esfuerzo y tiempo de desarrollo, por la dificultad y la experiencia que requiere trabajar con este tipo de aplicaciones [61] [16].

La aplicación móvil nativa es específica para cada sistema operativo móvil, y presenta las siguientes características [36]:

- **interfaz de usuario acorde al dispositivo:** cada sistema operativo es caracterizado por su propio estilo y modo de interacción, normalmente conocido como el *look and feel*[10];
- **acceso al sistema:** implica el acceso a aplicaciones ya instaladas, a la información contenida en el dispositivo (lista de contactos, galería de fotos, entre otros), y a escuchar los eventos y notificaciones del sistema (evolución del consumo de batería, recibo de un SMS, reanudación del sistema luego de una suspensión, entre otros);
- **acceso al hardware específico:** los teléfonos móviles inteligentes poseen una gran variedad de sensores (por ejemplo, sensores de luz, aceleración, presión, movimiento, elevación, localización, orientación del dispositivo) y recursos de hardware a los que se tiene acceso (por ejemplo, Bluetooth, NFC, conectividad, entre otros);
- **mejor rendimiento:** el mejor aprovechamiento de recursos de hardware y del sistema, y la apariencia acorde al dispositivo, permite al usuario final una mejor Experiencia de Usuario, del inglés *User Xperience* (UX). Además, el código fuente es eficiente, con alto rendimiento, consistente y con total acceso al hardware y datos subyacentes del dispositivo;
- **distribución en tiendas virtuales**, donde son ofrecidas al mercado para su comercialización [61].

El principal problema con este tipo de aplicaciones es lidiar con las diferentes plataformas disponibles: el desarrollo para cada una ellas incrementa el costo de mantenimiento y solicita cierto nivel de experiencia (implica un costo elevado tener que conocer todo el entorno propio de cada plataforma). Esto significa dificultad y costo en términos de esfuerzo y desarrollo para una única aplicación, estableciéndose así que el reto de la generación multiplataforma se encuentra en el desarrollo de aplicaciones nativas, y no tanto en los demás enfoques [23].

[7] Android Studio. http://developer.android.com/intl/es/tools/studio/index.html

[8] Xcode. https://developer.apple.com/xcode/

[9] Visual Studio. https://www.visualstudio.com/

[10] *Look and feel* es un término usado para describir la interfaz de usuario en términos de apariencia y función [16].

Para enfrentar estos problemas de desarrollo e interoperabilidad en móviles, una alternativa que también está siendo abordada es el desarrollo multiplataforma para ambientes web. Aun así, la aplicación nativa es estable, segura y más capaz de acceder a los recursos del dispositivo, y sigue siendo la elección para los desarrolladores de la mayoría de las aplicaciones sociales, juegos y de comunicaciones, brindando mayor UX [2].

2.2.2 Desarrollo de aplicaciones web

Una de las alternativas más utilizadas por las empresas es el desarrollo móvil multiplataforma, el cual permite simplificar el proceso de mantenimiento y despliegue, y ahorrar tiempo de desarrollo y esfuerzo [61]. El desarrollo de aplicaciones web implica usar un sólo código base para todas las plataformas, convirtiéndose en posible alternativa contra el problema de la fragmentación [48].

Estos tipos de aplicaciones no necesitan ser instalados en el dispositivo para poder ejecutarse ya que se ejecutan en un navegador. Se desarrollan usualmente usando PHP, Node.js, ASP.NET, HTML, CSS, o JavaScript [16] [59]. HTML5 es una alternativa prometedora: las aplicaciones basadas en esta tecnología, cuyo objetivo es unificar mediante la utilización de navegadores y componentes web, pueden correr en cualquier plataforma móvil sin ningún esfuerzo extra [59].

Las ventajas al trabajar con este tipo de aplicación son: i) reducción de los conocimientos requeridos para desarrollar aplicaciones, ya que se usa un lenguaje común; de la misma forma, el conocimiento del API propio de cada plataforma se reduce, ya que se manejan las comunes provistas por la herramienta; ii) reducción del código, debido a que se escribe una sola vez y se compila para cada plataforma soportada, lo que conlleva a una reducción del tiempo de desarrollo; iii) ampliamente soportado por la industria, de manera que la mayoría de los dispositivos tienen soporte para utilizarlo (sólo se necesita de un navegador) [61] [59].

Las desventajas, respecto a las aplicaciones nativas, son: i) menor rendimiento, puesto que todo se ejecuta mediante el intérprete JavaScript del navegador, cuya potencia es limitada; ii) tanto la interfaz del usuario (no se adapta a las características propias de la plataforma subyacente) como el acceso a recursos de hardware de los dispositivos (los sensores y recursos como cámara, Bluetooth, entre otros), son muy limitados; iii) no puede ser distribuida por tiendas virtuales, resultando complicada su comercialización [16].

Con las aplicaciones web se ahorran costos de esfuerzo y tiempo en el desarrollo multiplataforma, pero ofrecen una menor UX, y no hacen uso eficiente de los recursos del dispositivo. Ante esto, surgen aplicaciones que aprovechan lo mejor de estos enfoques de desarrollo: aplicaciones web móviles nativas o híbridas. A continuación veremos más detalles.

2.2.3 Desarrollo de aplicaciones web móviles nativas

Propiamente no son aplicaciones web ni tampoco nativas. Se ejecutan en un navegador, o mejor dicho, con un componente nativo que delega en un navegador (conocido como *native wrapper*). En otras palabras, no tienen la potencia de las aplicaciones nativas, simplemente ejecutan código en un navegador embebido (generalmente con HTML5). También conocidas como aplicaciones híbridas [59].

Este tipo de aplicación posee todas las ventajas de las aplicaciones web. Así mismo, en aspectos como instalación y distribución, se las puede considerar como aplicaciones nativas. De la misma manera, posee la mayoría de las desventajas de las aplicaciones web. En cuanto a UX, a pesar de tratarse de una aplicación nativa, requiere de conexión a Internet para poder trabajar, y funciona según el navegador [59].

2.3 El problema de la fragmentación

La variedad de teléfonos móviles inteligentes disponibles hoy día en el mercado es muy amplia. El número de sistemas operativos móviles no es tan elevado, pero las diferentes versiones de un mismo sistema operativo incrementan la variabilidad con la que se debe lidiar al momento de desarrollar aplicaciones. Cada sistema operativo móvil presenta un IDE propio, con un diseño de interfaz diferente; así también, el manejo de los recursos de hardware y de los datos en las aplicaciones suele realizarse de manera diferente en cada sistema [21]. Este ambiente fragmentado es uno de los principales retos para los desarrolladores de aplicaciones móviles en la actualidad, donde la variedad de plataformas origina este fenómeno conocido como **fragmentación** [59] [40] [51] [49].

La rápida proliferación de los diferentes sistemas operativos móviles ha obligado el desarrollo de aplicaciones específicas para cada teléfono inteligente, debido a que cada una de ellas posee un lenguaje de programación, SDK y API propio, es decir, tecnologías y herramientas a favor de facilitar la tarea del programador, ocasionando problemas de portabilidad entre estos sistemas. El esfuerzo de desarrollo se incrementa casi linealmente ante tan profundas diferencias [24]. Pero, con todo esto, incluso configurar el ambiente de desarrollo es una tarea compleja, por la variedad de IDE existentes y al hecho de tener que entender cómo usar la API exclusiva de cada plataforma [5]. Por este y otros motivos, existen factores que hacen más difícil el desarrollo de aplicaciones nativas para estos teléfonos en comparación a las aplicaciones de escritorio: variedad de gama de plataformas disponibles; variedad de dispositivos en términos de complejidad, tamaño, poder de cómputo, y a su vez; hardware e interfaces diferentes para cada tipo de dispositivo.

La fragmentación en Android. Como vimos en la figura 2.1, Android domina el mercado de los teléfonos móviles inteligentes en la actualidad. Es sin duda el sistema operativo móvil más popular, utilizada en más de 190 países, según la página oficial de desarrolladores para Android[11].

Android es uno de los principales afectados por la fragmentación. El motivo principal de esto es el deseo de Google, de que Android estuviera presente en el mayor número de teléfonos móviles, por lo cual lo hizo de código abierto, de modo que cualquier fabricante puede desarrollar terminales que incorporen dicho sistema operativo, recayendo sobre ellos la responsabilidad de hacer disponibles las nuevas versiones del sistema operativo.

El sitio oficial de Android[12] presenta datos muy interesantes sobre la situación en el mercado de las distintas versiones que podemos encontrar actualmente: parte de Froyo (API 8, versión 2.2) hasta Nougat, la última versión disponible (API 24, versión 7.0). Además, tanto Lollipop (API 22) como Marshmallow (API 23), ocupan los primeros puestos en las versiones más utilizadas entre los teléfonos inteligentes que utilizan Android. En total existen 11 versiones de API disponibles, distribuidas en 9 versiones diferentes.

La salida de nuevas versiones implica mantener las versiones antiguas, y brindar soporte de compatibilidad, para poder hacer uso de las nuevas funcionalidades de las APIs nuevas, mientras los dispositivos se actualizan. En Android se habla de soporte de diferentes dispositivos[13], donde se especifican técnicas para manejar distintos idiomas, tamaños de pantallas y versiones de la plataforma. También supone que el rendimiento de las aplicaciones sea distinto dependiendo de la versión de Android en la que se ejecute, o incluso el acceso a distintas características o funcionalidades del dispositivo. Se puede notar el esfuerzo que conlleva desarrollar aplicaciones nativas para esta plataforma.

De igual forma, la fragmentación influye en la seguridad de los usuarios, constituyendo un problema mayor. La mayoría de las vulnerabilidades que se identifican, únicamente son solucionadas para la última versión del sistema operativo, dejando en peligro a la mayor parte de los usuarios de la plataforma que aún no reciben (o que simplemente no recibirán, dependiendo del dispositivo) dichas actualizaciones. En el caso de iOS, este hecho es muy raro, puesto que en el momento en el que Apple lanza una nueva versión del sistema operativo, transcurre un breve período de tiempo hasta que la mayoría de sus dispositivos adoptan dicha versión (no todos los dispositivos reciben las últimas actualizaciones, pero normalmente son aquellos con una antigüedad que puede considerarse que han finalizado su vida útil, cosa que no sucede con los de Android).

[11] Acerca de Android. https://developer.android.com/about/android.html

[12] Informaciones acerca de Android. https://developer.android.com/about/dashboards/index.html

[13] Soporte de diferentes dispositivos Android. https://developer.android.com/training/basics/supporting-devices/index.html

2.4 Estudio sobre las alternativas para el desarrollo de aplicaciones móviles

El desarrollo de aplicaciones móviles cuenta con dos grandes enfoques de desarrollo: enfoque manual y enfoque (semi)automatizado.

Cada sistema operativo móvil cuenta con su IDE, brindando al desarrollador el ambiente, servicios y herramientas necesarios para poder crear aplicaciones, utilizando un lenguaje de programación específico. Ese tipo de desarrollo es totalmente manual, generando código fuente sin ningún nivel de abstracción ni automatización, pero con ayuda del IDE para facilitar el desarrollo (por ejemplo, detección de errores, funcionalidades del API con autocompletado, depuradores o *debugger*, entre otros). Las aplicaciones resultantes de este tipo de desarrollo pueden ser tanto nativas como web.

Por otra parte, existen otras alternativas que nos permiten desarrollar aplicaciones móviles, adoptando ciertos niveles de abstracción y (semi)automatizando ciertas partes del desarrollo. Hablamos de los *frameworks* y enfoques dirigidos por modelos [41].

La mayor ventaja de los *frameworks* es la reutilización de código y la rápida configuración para múltiples plataformas; pero, no reducen la complejidad de la implementación porque no proveen una visión general del sistema a un alto nivel, como sí sucede utilizando un enfoque dirigido por modelos. En lugar de eso, solo reducen el número de implementaciones requeridas (uno solo para varias plataformas), lo que no implica que sea más simple que usando un SDK nativo, y en algunos casos no producen una aplicación verdaderamente nativa [51]. Los *frameworks* actuales tratan de combinar los aspectos positivos de las aplicaciones web y nativas, mientras que con un enfoque dirigido por modelos se logra tener una visión más general y abstracciones en todo el conjunto de los subproblemas del dominio, mediante Lenguaje Específico del Dominio, del inglés *Domain Specific Language* (DSL) [41]. El problema con el DSL es su poca documentación, dificultando su aprendizaje, pero la ventaja es que presenta construcciones dedicadas a un dominio (en nuestro caso, dominio móvil); y mediante los modelos, obtenemos un nivel de abstracción más elevado, pudiendo generar a partir de ellos (semi)automáticamente la aplicación [36].

Ambos enfoques, como vimos, presentan ventajas y desventajas. En las siguientes secciones analizaremos qué herramientas disponibles existen, y discutiremos características y aspectos resaltantes sobre los mismos.

2.4.1 Contexto del estudio

Para entender las tendencias actuales de desarrollo, se llevó a cabo un estudio de revisión de *frameworks* y propuestas dirigidas por modelos. Para el estudio de los *frameworks* nos basamos en trabajos como [24] [48] [55] [45] [26] [14] [49] [36], que evalúan y comparan varias de estas herramientas; y de igual forma, investigamos en las páginas oficiales de estos. Por otra parte, las propuestas dirigidas por modelos resultan de un proceso de investigación de la literatura, disponibles en librerías digitales como *IEEE Xplore*[14], *SpringerLink*[15] y *Google Scholar*[16]. El objetivo de esta búsqueda fue obtener resultados preliminares que nos permitan emitir juicios iniciales sobre la adopción de enfoques dirigidos por modelos para el desarrollo de aplicaciones móviles. Los criterios de selección de todos estos trabajos se basaron en: i) que soporte al menos la generación de aplicaciones para dos plataformas (para así poder verificar la generación multiplataforma); y, ii) que se encuentre disponible o en proceso de desarrollo, y en constante mantenimiento (nos interesan solamente las herramientas actuales).

La información recabada de cada herramienta está basada en los siguientes aspectos: el tipo de licencia que contempla, si es de código abierto o no, el estado de la herramienta (disponible o trabajo en desarrollo), los sistemas operativos que soporta, el lenguaje de programación que utiliza, si permite modelado y de qué tipo, el tipo de aplicación que genera (nativo, web o híbrido); y por último, si permite el acceso al API nativo de la plataforma (indica si puede acceder a recursos del dispositivo).

De los estudios mencionados obtuvimos una lista de *frameworks* y propuestas de desarrollo dirigido por modelos. A continuación hablamos de los aspectos más destacados encontrados.

2.4.2 Discusión sobre resultados

Como resultado del estudio llevado a cabo, contamos con las siguientes 13 (trece) alternativas para desarrollar aplicaciones móviles: 6 (seis) *frameworks* y principalmente, 7 (siete) propuestas dirigidas por modelos. Entre los *frameworks* actualmente disponibles en el mercado, encontramos que los más populares son Apache Cordova[17] y Titanium[18] [41]. De

[14] IEEE Xplore. http://ieeexplore.ieee.org/Xplore/home.jsp
[15] SpringerLink. http://link.springer.com/
[16] Google Scholar. https://scholar.google.com/
[17] Apache Cordova. https://cordova.apache.org/
[18] Titanium. http://ow.ly/vKf1307urEH

igual forma, encontramos a Rhodes[19], Sencha[20], IBM Mobile First[21] y Xamarin[22]. Por otra parte, están las propuestas dirigidas por modelos, en su mayoría DSL como: MD2 [39], Mobia [4], XIS-Mobile DSL [51], AXIOM DSL [30], Xmob DSL [36], MobL [25] y plataformas disponibles como WebRatio móvil[23] [1]. Basados en los aspectos recabados para cada herramienta y/o propuesta, a continuación brindamos una discusión sobre cada uno de estos aspectos.

» Respecto al enfoque de desarrollo utilizado

El desarrollo nativo es una mayoría entre los enfoques de desarrollo que ofrecen las herramientas. Tenemos un total de 9/13 (69%) herramientas que permiten generar aplicaciones nativas. Entre estos, 4/6 (67%) constituyen *frameworks* y 5/7 (71%) son DSL. Entre los *frameworks*, el desarrollo de aplicaciones híbridas constituyen la opción dominante de generación.

Una característica importante del desarrollo nativo de aplicaciones, y que lo diferencia de otros enfoques, es poder acceder a los recursos de cada dispositivo (al API de la plataforma). Entre las propuestas dirigidas por modelos destacamos que todas soportan el acceso a los recursos del dispositivo, excepto Mobia, que no encontramos mayor información al respecto. Por otra parte, entre los *frameworks*, Rhode, Sancha y Cordova permiten el acceso a ciertos recursos mediante JavaScript. Xamarin también permite el acceso mediante C#.

» Respecto a los lenguajes de desarrollo utilizados

Ningún lenguaje utilizado en los *frameworks* requiere un esfuerzo importante para su aprendizaje, puesto que son lenguajes muy adoptados para el desarrollo, como C#, .NET y las tecnologías web como HTML5, CSS y JavaScript. Entre los lenguajes de desarrollo más utilizados tenemos a HTML5 en un 3/6 (50%) de los *frameworks*. Entre las propuestas dirigidas por modelos sí existe ese esfuerzo de aprender un lenguaje nuevo de modelado, tenemos por ejemplo a MD2, Xmob, Axiom y XIS-Mobile: todos DSL específicos para el dominio móvil. No pasa así con Rhodes y Mobia, que requieren de HTML5, CSS y JavaScript para el modelado. Resalta que solamente XIS-Mobile DSL y WebRatio proponen la extensión de un lenguaje de modelado ya existente, en lugar de introducir uno nuevo (lenguajes XIS y IFML respectivamente).

[19] Rhodes. http://rhomobile.com/
[20] Sencha. https://www.sencha.com/
[21] IBM Mobile First. http://www.ibm.com/mobilefirst/us/en/
[22] Xamarin. https://xamarin.com/
[23] WebRatio móvil. http://ow.ly/sNFT307urHd

» Respecto a los sistemas operativos móviles soportados

Android y iOS resultan ser los sistemas operativos más soportados tanto por los *frameworks* como las propuestas dirigidas por modelos.

» Respecto al estado y licencia de la herramienta

Como vimos, existen varias soluciones para el desarrollo multiplataforma. La mayoría son herramientas disponibles, a excepción de MD2, XMOB, AXIOM y XIS-Mobile que están en fase de desarrollo. Entre éstas, solamente Rhodes, Titanium y Apache Cordova son código abierto. Entre las propuestas dirigidas por modelos, no encontramos mucha información al respecto.

2.4.3 Puntos resaltantes obtenidos

Si bien, este estudio tuvo como objetivo obtener datos iniciales sobre las herramientas y propuestas disponibles para el desarrollo de aplicaciones móviles, nos sirvió para poder emitir juicios iniciales al respecto. De la discusión de cada uno de los aspectos recabados, resaltamos los siguientes puntos:

- son más las alternativas basadas en modelos que permiten generar aplicaciones verdaderamente nativas, que los *frameworks* (éstos están más enfocados en el desarrollo de aplicaciones híbridas);
- las propuestas dirigidas por modelos permiten el acceso a API nativo, ya que sus DSL son específicos del dominio móvil;
- los *frameworks* no requieren el aprendizaje de nuevos lenguajes de programación, ya que utilizan tecnologías y lenguajes web muy utilizadas actualmente (por ejemplo, HTML5, JavaScript y CSS); en cambio, la mayoría de las propuestas dirigidas por modelos proponen lenguajes nuevos que requieren cierto aprendizaje;
- por lo general, la mayoría de los *frameworks* disponibles ya son herramientas robustas, disponibles y con buena documentación; los DSL en su mayoría, son propuestas aún en desarrollo y con poca documentación (por lo tanto, poco accesibles);
- los sistemas operativos más soportados son Android y iOS.

Los frameworks se enfocan más en el desarrollo de aplicaciones híbridas, y permiten la reutilización de código y la configuración rápida para múltiples plataformas, pero no proveen una visión general del sistema a un alto nivel, como lo hacen los enfoques dirigidos por

modelos [41]. Teniendo en cuenta lo mencionado previamente, para el desarrollo nativo de aplicaciones móviles, creemos que los enfoques dirigidos por modelos podrían constituirse en opción factible y adecuada contra la fragmentación. Este enfoque nos permite trabajar en el dominio de los móviles, con construcciones dedicadas a este dominio; y mediante los modelos, obtenemos un nivel de abstracción más elevado, pudiendo generar a partir de ellos (semi)automáticamente la aplicación verdaderamente nativa [36]. En el capítulo 4 hablaremos en detalle sobre el Desarrollo Dirigido por Modelos (MDD): analizaremos algunas de sus principales características, y específicamente veremos cómo encaja este enfoque en el desarrollo de aplicaciones móviles, observando en mayor detalles qué propuestas encaran este proceso.

2.5 Síntesis del capítulo

Los teléfonos móviles inteligentes cuentan con avanzados recursos computacionales, permitiendo correr aplicaciones bajo un avanzado sistema operativo. Android se destaca entre los sistemas operativos más utilizados actualmente.

Las aplicaciones móviles manejan sensores y hardware específicos del teléfono, y trabajan con recursos limitados como memoria, batería, conectividad, entre otros. Los recursos limitados del teléfono móvil son aspectos importantes a tener en cuenta al momento de desarrollar una aplicación móvil. En particular, destacamos la conectividad. Generalmente, los teléfonos inteligentes requieren de una conexión activa a Internet para realizar peticiones a servicios web y procesar las respuestas. En caso que la red no esté disponible, es necesario contar con almacenamiento local (en modo *offline* o "sin conexión"). La arquitectura tradicional de una aplicación es la multicapas: capa de presentación, de negocios y de datos.

El ambiente fragmentado es uno de los principales retos para los desarrolladores de aplicaciones móviles en la actualidad, donde la variedad de plataformas origina este fenómeno conocido como fragmentación. El reto de la generación multiplataforma se encuentra en el desarrollo de aplicaciones nativas, donde desarrollar para cada plataforma implica esfuerzo, costo de mantenimiento y requiere de cierto nivel de experiencia.

El estudio que llevamos a cabo nos permitió identificar enfoques (semi)automatizados, que proveen gran ayuda para el desarrollo de aplicaciones móviles: *frameworks* y enfoques dirigidos por modelos. Si bien, este estudio fue carácter general, nos permitió identificar varias herramientas y propuestas dirigidas por modelos. Una comparación entre estos, nos indica que un enfoque dirigido por modelos podría constituirse en una opción factible y adecuada contra la fragmentación, al estar más orientado al desarrollo de aplicaciones nativas que los *frameworks*, específicamente en el dominio de los móviles.

Capítulo 3

La persistencia de datos

La conectividad constituye un recurso limitado para los teléfonos móviles (sección 2.1). La variabilidad de la conexión, o fenómeno conocido como naturaleza transitoria de los teléfonos móviles [16], implica una conexión a red en constante variación, e incluso, en la pérdida o caída de conexión. Para aquellas aplicaciones móviles que requieren de una conexión activa a Internet para realizar peticiones a servicios web y procesar las respuestas, contar con almacenamiento local es necesario en caso que no se cuente disponible la red.

En este capítulo se presenta la persistencia de datos en los teléfonos móviles inteligentes, destacando y analizando el problema asociado y los distintos mecanismos de persistencia existentes.

En la sección 3.1 se presenta el problema relacionado a las limitaciones en la conectividad, y se describe la importancia de la persistencia en esas situaciones.

Para entender en detalle cómo es el almacenamiento en los teléfonos móviles, en la sección 3.2 se analiza las distintas opciones disponibles para almacenar datos. Además, se estudia cómo afecta el problema de la fragmentación a la persistencia de datos en aplicaciones móviles, viendo que cada sistema operativo móvil maneja de distintas formas las opciones de almacenamiento. Y finalmente, se habla sobre unos de los tipos de almacenamientos más conocidos y utilizados: base de datos integradas en los teléfonos móviles.

La arquitectura por capas implica un agrupamiento lógico de componentes separado en capas que se comunican entre sí y con otros clientes y aplicaciones [46]. Como vimos en la sección 2.1.2, esta es por lo general la arquitectura típica de las aplicaciones móviles. En la sección 3.3 se finaliza hablando en detalle sobre la capa de datos y los aspectos principales que cubre: la persistencia de datos y el manejo de datos provenientes de distintas fuentes.

3.1 El problema de la persistencia

El acceso a datos en los dispositivos móviles es restringido por las conexiones de red no confiables y por las limitantes de hardware del dispositivo (como la memoria, batería, conectividad, entre otros). No considerar aspectos como bajo ancho de banda, alta latencia, y la intermitencia de la conectividad al momento de diseñar el acceso de datos, puede afectar dicho diseño [8] [40] [27] [46]. Este fenómeno es conocido como la **naturaleza transitoria** de los teléfonos móviles inteligentes [16].

La variabilidad de la conexión se define y manifiesta en dos grandes preocupaciones: i) los dispositivos móviles, en especial los teléfonos móviles inteligentes, poseen conectividad de red cambiante (redes 3G, 4G y *wireless*, entre otros), en constante variación, casi imperceptibles para el usuario; ii) la pérdida o caída inesperada de conexión. La conectividad de red juega un papel crucial en aquellas aplicaciones que requieren traspaso de datos entre el dispositivo y un servidor: no tener en cuenta estos aspectos puede resultar en aplicaciones con poca usabilidad. Para mantener la UX y hacer frente a estos problemas, es necesario almacenar datos en caché y guardar información de forma local en el dispositivo [8].

Ante este panorama, las aplicaciones móviles utilizan el almacenamiento local de sus datos en caso que la red no esté disponible. MAAG recomienda usar el almacenamiento en caché para mejorar el rendimiento y *responsiveness* de la aplicación, y para permitir operaciones cuando no haya conexión a red [46].

Hoy en día tenemos a disposición una inmensa cantidad de aplicaciones de distintos tipos. Muchas de ellas son afectadas por la naturaleza transitoria de la conectividad. En la figura 3.1, resaltamos los casos de aplicaciones muy conocidas como Netflix[1], Spotify[2], Google Translate[3] y Youtube[4], en sus versiones para teléfonos móviles inteligentes, que son afectados por la variabilidad de la conectividad, donde la pérdida de conexión implica una UX afectada, imposibilitando acceder al contenido de la aplicación. Cabe resaltar, que nuevas actualizaciones de estas aplicaciones están permitiendo descargar cierto contenido específico al teléfono, para tenerlo disponible cuando no haya conexión (el acceso es exclusivamente al contenido descargado): Google Translate, permite descargar paquetes de idiomas, y por otra parte, una de las últimas actualizaciones de Netflix, permite descargar cierto contenido (series y películas) para poder verlas aún sin conexión de red. Es notable que cada vez más empresas están adoptando esta modalidad de trabajo, conscientes de que no contar con mecanismos de

[1] Netflix. http://netflix.com/
[2] Spotify. https://www.spotify.com/py/
[3] Google Translate. https://translate.google.com/
[4] Youtube. https://www.youtube.com/

persistencia, que aseguren la UX aún en condiciones sin conexión de red, significa pérdida para ellos.

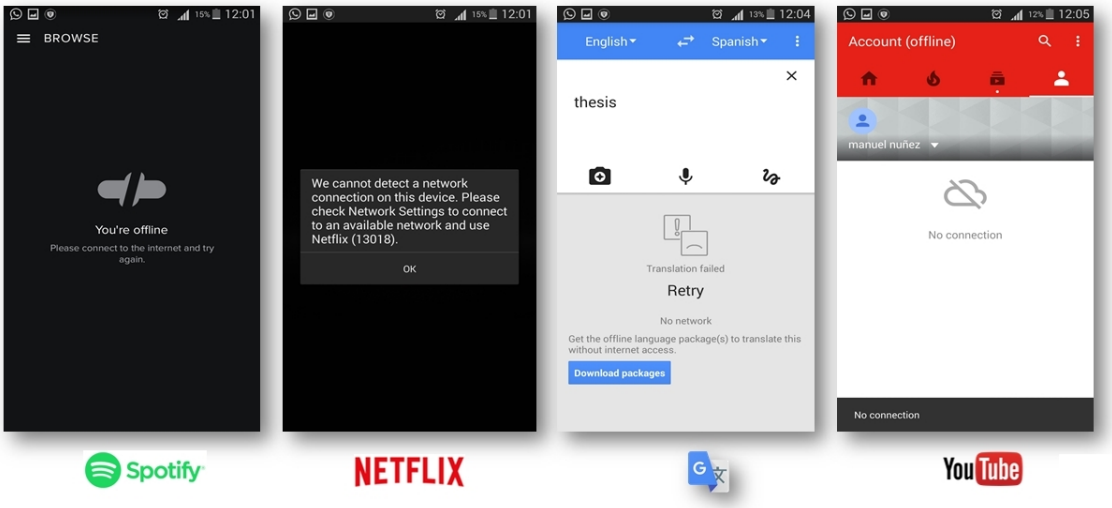


Figura 3.1 Vista de aplicaciones conocidas como Spotify, Netflix, Google Translate y Youtube, sin conexión a red

El almacenamiento de datos en los teléfonos móviles

El desarrollo de aplicaciones móviles generalmente involucra guardar datos. Estos datos pueden ser de distintos tipos: desde imágenes provenientes de la cámara, a configuraciones de preferencia del usuario, respaldo de archivos, documentos, entre otros. Estos datos pueden provenir de una fuente externa (por ejemplo, un servidor remoto), los cuales pueden ser manipulados localmente; o bien, pueden provenir del mismo teléfono (por ejemplo, los sensores). Este tipo de aplicaciones son conocidas como aplicaciones dirigidas por datos (o *data driven apps*). El nivel de aplicación varía si los datos se persisten o simplemente se mantienen en la memoria, puesto que los teléfonos móviles tienen limitación de ancho de banda y espacio, lo que requiere que ciertos datos deban ser guardados en caché (almacenados localmente pero no permanentemente, con la opción de que puedan ser sincronizados luego) [40] [27]. Esto da lugar al modo "sin conexión" que muchas aplicaciones ofrecen, para el cual es necesario guardar en caché ciertos datos necesarios para que la aplicación funcione sin conexión a red [41]. Aquí radica la importancia de contar con bases de datos integradas, como SQLite, u otros mecanismos que nos proveen los sistemas operativos.

En la figura 3.2, presentamos el caso de la aplicación de mensajería Whatsapp[5], cuyo sistema de archivos nos permite identificar la utilización de distintas opciones de mecanismos

[5]Whatsapp. https://www.whatsapp.com/?l=es

de almacenamiento: base de datos interna y documentos guardados (media). Whatsapp es una aplicación de mensajería que no maneja simplemente texto, sino también media (audio, imágenes y vídeos), localización, datos del dispositivo (por ejemplo, lista de contactos). Este contenido se queda almacenado en el teléfono, disponible para el usuario aun cuando no se cuente con una conexión a red. Inclusive, las conversaciones quedan guardadas, de manera que uno pueda acceder a ellas también. Esta aplicación se encarga de brindar una UX estable e ininterrumpida bajo cualquier circunstancia.

La persistencia de datos en los teléfonos móviles inteligentes es una realidad que cada vez más impone su importancia en la fase de desarrollo de aplicaciones. Cada vez más se necesita manejar y tener disponibles datos en diversas situaciones. A nivel código, la cantidad requerida para el manejo de todas las operaciones constituye una porción significante de la aplicación entera. Alrededor del 21% del código de la aplicación se destina a las conexiones, a los modelos de datos y a la configuración de las distintas fuentes de datos (tanto para el almacenamiento local como para recibir e intercambiar datos con fuentes externas). Esta cantidad de código implica una cantidad de esfuerzo para desarrollarlo, y sería costoso realizar cambios posteriores, por lo que tener en cuenta la persistencia en etapas tempranas de desarrollo sería importante [41]. Para esto, es necesario llevar a cabo un análisis del modelo de datos a utilizar en la aplicación, así mismo, conocer y estar al tanto de los distintos mecanismos de persistencia disponibles a fin de utilizarlos adecuadamente.

A continuación se detallan las distintas opciones de almacenamiento disponibles actualmente para los teléfonos móviles inteligentes.

Figura 3.2 Mecanismos de persistencia utilizado por Whatsapp

3.2 Análisis del almacenamiento en los teléfonos móviles

Para poder manejar el acceso a datos en la aplicación móvil, es necesario conocer con qué opciones de almacenamiento disponemos, y de esa manera poder escoger el adecuado para la aplicación. En la sección 3.2.1 veremos cuáles son estas opciones, y hablaremos sobre ciertos mecanismos de persistencia que se derivan de estas opciones y son comunes para los distintos dispositivos.

Por otra parte, en la sección 3.2.2 analizaremos una cuestión interesante: la fragmentación y los mecanismos de persistencia. Realizamos un estudio sobre el manejo de los distintos mecanismos de persistencias en diferentes sistemas operativos móviles.

Por último, en la sección 3.2.3 hablaremos en detalle sobre las base de datos móviles, uno de los mecanismos de persistencia más utilizados.

3.2.1 Opciones más comunes de almacenamiento en la actualidad

Mahmoud et al. [38] y Fotache et al.[17] señalan las opciones de software más comunes para almacenamiento de datos en las aplicaciones móviles, en las distintas plataformas móviles:

- **HTML5**: permite el desarrollo de aplicaciones multiplataforma usando tecnologías web. Provee de un API para almacenamiento, siendo el intérprete JavaScript del navegador el intermediario. Algunos métodos de almacenamiento son:
  - Almacenamiento de objetos por aplicación o por dominio web (*Local Storage* y *Global Storage*). El más utilizado es el *localstorage API*, el cual permite almacenar pares clave – valor, usando cadenas. Con esta opción no es posible almacenar construcciones de datos muy complejas, pero existe la opción de almacenar datos en forma de cadena con el método JSON.stringify.
  - IndexedDB, el cual provee una implementación de una base de datos relacional a la aplicación, con la posibilidad de hacer consultas SQL.

  Así mismo, HTML5 provee ahora la modalidad "sin conexión" para la aplicación (utilizado para geolocalización, gráficos canvas y reproducción de audio/vídeo) lo que permite correr las aplicaciones web cuando no hay una conexión de red activa, y para almacenar bases de datos persistentes en el propio navegador web (por ejemplo, Gmail[6] de Google) [16].

[6]Gmail. https://mail.google.com/mail/u/0/#inbox

Uno de los aspectos a tener en cuenta con HTML5, al momento de utilizarlo en el desarrollo, es que las diferentes plataformas utilizan diferentes motores en los navegadores para implementar esta tecnología.

- **Almacenamiento en la nube**: es una amplia colección de servicios de alojamiento de archivos como Apple iCloud[7], Dropbox, Google Drive, Mega[8], Amazon S3[9], Ubuntu Cloud[10], entre otros.

  Estos servicios proveen una gran cantidad de espacio accesible, pero la latencia para su acceso es relativamente alta en comparación con el acceso local. En los teléfonos móviles se utilizan llamadas asíncronas para paliar este problema.

- **SQLite**: es una de las opciones de almacenamiento más populares disponibles en el mercado en la actualidad. Constituye una de las bases de datos para móviles con más atractivo para los desarrolladores: es simple, multiplataforma, fácil de instalar y configurar, no requiere de un servidor, es compacto, portable y de dominio público. Comprende una librería que encapsula funcionalidades SQL. Se instala localmente en el teléfono en lugar de una conexión a una base de datos remota.

  Muchas aplicaciones en distintas plataformas utilizan SQLite para almacenar datos estructurados en una base de datos privada, desde perfiles de usuario a ajustes de configuraciones. Además, resulta muy adecuado para propósitos de testeo y desarrollo.

Las opciones presentadas podemos clasificarlas en:

- **almacenamiento local**: entre los mecanismos de almacenamiento locales más comunes encontramos:
  - pares clave-valor, también conocidos como diccionarios y listas;
  - archivos y documentos (o *files*);
  - almacenamiento en dispositivos externos (por ejemplo, tarjetas de memoria SD);
  - bases de datos integradas;
- **almacenamiento remoto**: contamos con el almacenamiento en la nube (ya sea en servicios web ofrecidos por empresas, o base de datos alojadas en servidores remotos).

[7] iCloud. https://www.icloud.com/
[8] Mega. https://mega.nz/
[9] Amazon S3. https://aws.amazon.com/es/
[10] Ubuntu Cloud. https://www.ubuntu.com/cloud

3.2.2 Manejo en distintos sistemas operativos móviles

	Mecanismos de persistencia			
Plataforma	**Par clave-valor**	**Manejo de archivos**	**Soporte para almacenamiento externo**	**Base de datos**
Android	SharedPreferences, Bundles	Java Files Stream	Sí	Content Provider, SQLite
iOS	NSUserDefaults, Property lists, NSCoding	NSFileManager	Sí	Core Data, SQLite
Windows Phone	Isolated Storage Settings	Isolated Storage File System	Sí	Isolated Storage, SQLite

Cuadro 3.1 Mecanismos de persistencia de datos en sistemas operativos móviles

Todas las opciones de almacenamiento mencionadas tienen soporte en los distintos sistemas operativos móviles; pero, es importante notar que en cada una de ellas, existen variaciones en el manejo de estas opciones. En el cuadro 3.1 resumimos cómo manejan los sistemas operativos móviles más populares las distintas opciones de persistencia [21] [38]. A continuación hablaremos un poco más en detalle de cada una de estas plataformas:

- **Android**[11]: cuando hablamos de guardar estados de la aplicación, opciones de configuración de la aplicación y datos del usuario, Android cuenta con las Preferencias Compartidas o *SharedPreferences*. Este almacena en espacio de memoria no-volátil, datos privados en pares clave-valor y actúa como una tabla liviana de *hash* que persiste datos entre aplicaciones. Soporta los tipos de datos primitivos como: *Boolean*, *Float*, *Integer*, *Long* y *String*. Muy relacionados a esta forma de almacenamiento y parecidos a los diccionarios, se encuentran los *Bundles*, utilizados principalmente para el almacenamiento de estados de la aplicación, opciones de configuración, simple información del usuario y otros datos del usuario, así como paso de datos entre actividades (pantallas del sistema).

  La librería *Java Streams* permite a Android el manejo de lectura y escritura de archivos. Los datos de la aplicación son guardados en los archivos del sistema, generalmente en una ubicación por defecto para el almacenamiento de archivos y bases de datos, constituyendo el *Internal Storage* del dispositivo. Para el almacenamiento externo, todos los dispositivos Android cuentan con soporte de dispositivos de almacenamiento removibles, como por ejemplo, tarjetas de memoria SD.

  Para una solución de almacenamiento más sólida y para almacenar registros localmente, Android propone el uso de bases de datos. Para la misma cuenta con soporte para

[11] Android, opciones de almacenamiento . https://developer.android.com/guide/topics/data/data-storage.html?hl=es

SQLite. Estos datos por defecto son de carácter privado para otras aplicaciones, los *Content Providers* proveen un mecanismo de encapsulamiento y de seguridad de los datos, por medio del cual otras aplicaciones pueden acceder a los datos almacenados.

- **iOS**[12]: para recolectar preferencias del usuario, iOS cuenta con los *User Defaults*, los cuales actúan como diccionario, almacenan pares clave-valor. Es la parte del sistema que almacena y retorna las preferencias del usuario, implementada por el *NSUserDefaults* y permite almacenar tipos de datos básicos en el disco: *NSString*, *NSNumber*, *NSDate*, *NSArray* y *NSDictionary*. Los *Property lists*, utilizados para almacenar datos más simples; y el protocolo NSCoding, para almacenar objetos y estructuras, también persisten datos como pares clave-valor.

  De igual forma, iOS soporta el manejo de archivos con la clase *NSFileManager*, el cual permite crear y manipular archivos. Los desarrolladores de aplicaciones para iOS también pueden usar archivos XML para guardar configuraciones de las aplicaciones y preferencias del usuario.

  Para almacenar múltiples datos y más complejos, se utilizan bases de datos. Esta plataforma cuenta con *Core Data Framework*, el cual es más que una simple base de datos relacional, presentando facilidades al desarrollador al momento de trabajar con los modelos. De igual forma, iOS cuenta con SQLite como alternativa de uso para base de datos.

- **Windows Phone**[13]: para almacenar grandes cantidades de datos, la plataforma utiliza el *Isolated Storage File System* el cual permite almacenar datos en archivos, incluso usando carpetas para formar un directorio estructurado. Por otra parte, cuando el programa solamente necesita almacenar un simple valor, comúnmente se utiliza *Isolated Storage Settings*, el cual funciona como un diccionario, almacenando cualquier cantidad de pares clave-valor.

  El almacenamiento de datos estructurados se hace por medio de bases de datos integradas al teléfono. SQLite es una de las bases de datos disponibles, de amplio uso y de soporte en esta plataforma.

  A partir del Windows Phone 8 se cuenta con soporte para las tarjetas de memoria SD, constituyendo así un medio alternativo externo para almacenamiento de archivos.

Según la página oficial de desarrollo de iOS[12], *"la persistencia de datos es uno de los problemas más importantes y comunes en el desarrollo de aplicaciones iOS. iOS tiene*

[12] iOS, datos persistentes. http://ow.ly/5slZ307Lwzy

[13] *Data for Windows Phone 8*. https://msdn.microsoft.com/en-us/library/windows/apps/ff402541(v=vs.105).aspx

muchas soluciones de almacenamiento de datos persistente". Esto resume lo que vimos previamente, y lo mismo sucede con Android, Windows Phone y otros sistemas operativos: *ante tanta variedad de soluciones, cada plataforma adopta estos mecanismos según funciones específicas; notándose así, el problema que ocasiona la fragmentación en este aspecto en particular*.

3.2.3 Base de datos móviles

En la actualidad, la mayoría de las aplicaciones y servicios (juegos, *m-commerce* o comercio móvil, *e-commerce* o comercio electrónico y aplicaciones dirigidas por datos) necesitan tener soporte con bases de datos. Este ambiente de computación móvil a menudo involucra un sistema formado por la comunicación entre una base de datos central y la base de datos móvil, involucrando tareas como sincronización y consultas en ambas direcciones (de la base de datos central a la base de datos móvil, y viceversa). Las aplicaciones móviles no necesitan estar conectadas a la base de datos central si la información ya está almacenada en el teléfono; de allí radica la importancia de las bases de datos móviles [27].

Las bases de datos móviles integradas constituyen uno de los mecanismos de persistencia de datos más utilizados, son una forma de almacenamiento en caché. Estas bases de datos residen localmente en dispositivos como los teléfonos móviles inteligentes, computadoras portátiles, tabletas, entre otros. Están diseñadas para trabajar en un ambiente donde los recursos son limitados (memoria, batería, capacidad de computación) y de constante movilidad y desconexiones. Son capaces de comunicarse con una base de datos central u otros dispositivos móviles remotos. Una de sus principales tareas, además de almacenar datos del usuario y de la aplicación (por ejemplo, datos de un formulario, datos de inicio de sesión, datos de configuración de la aplicación) y guardar multimedia (imágenes, música, vídeo), es permitir el respaldo de archivos y de otras configuraciones, que permiten el manejo de *queries* de forma local y sin conexión [27] [44].

Las principales ventajas de usar bases de datos móviles son [63]:

- Acceso de datos sin conexión, donde el usuario puede leer y actualizar datos sin necesidad de una conexión de red.

- Supera problemas como pérdida de conexión, bajo ancho de banda, alta latencia y otros problemas comunes con las redes *wireless*. El no usar conexión de red en todo momento, supone un incremento de la vida de la batería del teléfono, así como una reducción en la tarifa de conexión inalámbrica (servicio de conexión 3G, por ejemplo).

- Con la sola necesidad de sincronizar los datos actualizados, se libera una cantidad innecesaria de intercambio de datos entre la base de datos móvil y la central, reduciendo el tráfico y mejorando la velocidad de uso de la aplicación móvil.

Los principales problemas a enfrentar en un ambiente de computación móvil son [63][27]:

- ***Responsiveness***, o la expectativa que se tiene de las aplicaciones móviles de que provean acceso a la información y servicios en cualquier momento y lugar. Como el dispositivo tiene almacenamiento y recursos limitados, es necesario que sean usados efectiva y eficientemente. Posibles soluciones a este problema. Identificar datos relevantes, algunas tablas en la base de datos central no son necesarias en la base de datos móvil, quizás solo algunas columnas o filas. Este acceso y búsqueda de estos datos podría significar una gran latencia en el acceso, por lo que se podría plantear que los datos estén organizados de manera que los datos relacionados sean puestos juntos. Otra propuesta es mejorar el cacheo, de manera a mejorar la taza de acierto. Mostrar y hacer consultas a la base de datos de los datos relevantes puede mejorar la *responsiveness*; así mismo, un preprocesamiento de los datos junto con una desnormalización podría mejorar el rendimiento en operaciones *joins* en múltiples tablas. Por último, cuando se quiera transferir grandes cantidades de datos, técnicas de compresión de datos pueden ser usadas.

- **Consistencia de datos y concurrencia**. Se espera que los dispositivos móviles trabajen adecuadamente, ya sea cuenten con una conexión a red o no. Se espera que las transacciones puedan ser manejadas en cualquiera de los casos mencionados, así como las desconexiones no planeadas. Es necesario que mecanismos de sincronización robustos mantengan actualizados ambas base de datos (la central y la del móvil).

- **Sincronización y resolución de conflictos**. La clave de la sincronización es mantener una imagen real de los datos entre todos los dispositivos móviles y la base de datos central. La sincronización es un componente crítico. Esta capa tiene que ser capaz de manejar no solo actualizaciones, sino también inserciones y borrados. También, tiene que actuar como mediador para evitar posibles conflictos luego de que ocurran cambios por cada usuario en el sistema. Mantener un conjunto de datos separados por cada usuario podría ser una forma de encarar este problema: pero esto no siempre es una opción. En esos casos, prioridades de sincronización pueden ser dados a cada usuario del sistema, de manera a evitar posibles solapamientos de datos.

- **Seguridad**. Autenticación de los usuarios, políticas de acceso (roles de acceso) y encriptación de datos podrían establecerse como medidas para asegurar que las

informaciones personales y/o criticas del usuario móvil sean resguardadas y mantenidas seguramente.

- **Alta disponibilidad**. Estrategias como planificación de respaldo automático, recuperación automática en el caso de fallas, así como replicación de datos, son algunas de las técnicas que pueden encargarse de asegurar una alta disponibilidad del sistema.

Base de datos	Tipo	Tipo de datos almacenados	Licencia	Plataformas soportadas
BerkeleyDB	relacional / NoSQL	objetos, pares clave-valor, documentos	AGPL 3.0[14]	Android, iOS
Couchbase Lite	NoSQL	documentos	Apache 2.0[15]	Android, iOS
LevelDB	NoSQL	pares clave-valor	New BSD[16]	Android, iOS
SQLite	relacional	NULL, Real, Entero, Texto, Blob	Dominio público	Android, iOS, Windows Phone, Blackberry
UnQLite	NoSQL	pares clave-valor, documentos	BSD 2-Clause[17]	Android, iOS, Windows Phone

Cuadro 3.2 Comparación de bases de datos móviles integradas actuales

Como observamos en el cuadro 3.2, las bases de datos integradas móviles más populares según[18] son: Berkeley DB[19], Couchbase Lite[20], LevelDB [21], SQLite y UnQLite[22]. Entre estas resalta SQLite, el cual es el motor de base de datos más utilizado actualmente[23], compatible y soportado por distintos sistemas operativos móviles como Android, iOS y Windows Phone. Su código fuente es de dominio público.

3.3 Capa de datos: persistencia y proveedores de datos

La capa de datos provee acceso a los datos del sistema (mediante interfaces genéricas expuestas que pueden ser consumidas), y a los datos expuestos por otros sistemas en red, quizás por medio de servicios. Como vemos en la figura 3.3, esta capa posee componentes tales como los Componentes de Acceso a Datos (o *Data Access Components*) quienes proveen

[14] AGPL 3.0. http://www.gnu.org/licenses/agpl-3.0.html
[15] Apache 2.0. http://www.apache.org/licenses/LICENSE-2.0.html
[16] New BSD. http://opensource.org/licenses/BSD-3-Clause
[17] BSD 2-Clause. http://opensource.org/licenses/BSD-2-Clause
[18] *Five popular databases for mobile*. https://www.developereconomics.com/five-popular-databases-for-mobile
[19] Berkeley DB. http://www.oracle.com/technetwork/database/database-technologies/berkeleydb/overview/index.html
[20] Couchbase Lite. https://www.couchbase.com/nosql-databases/couchbase-mobile
[21] LevelDB. https://github.com/google/leveldb
[22] UnQLite. http://unqlite.org/
[23] SQLite, *Most Widely Deployed and Used Database Engine*. https://sqlite.org/mostdeployed.html

las funcionalidades para acceder a los datos del sistema (datos almacenados), y componentes del Agente de Servicio (o *Service Agents*) quienes proveen las funcionalidades para acceder a los datos expuestos por otros sistemas *backend* por medio de servicios web. Adicionalmente también puede contener componentes que proveen funciones auxiliares y utilidades (o *Data Helpers / Utilities*) [46].

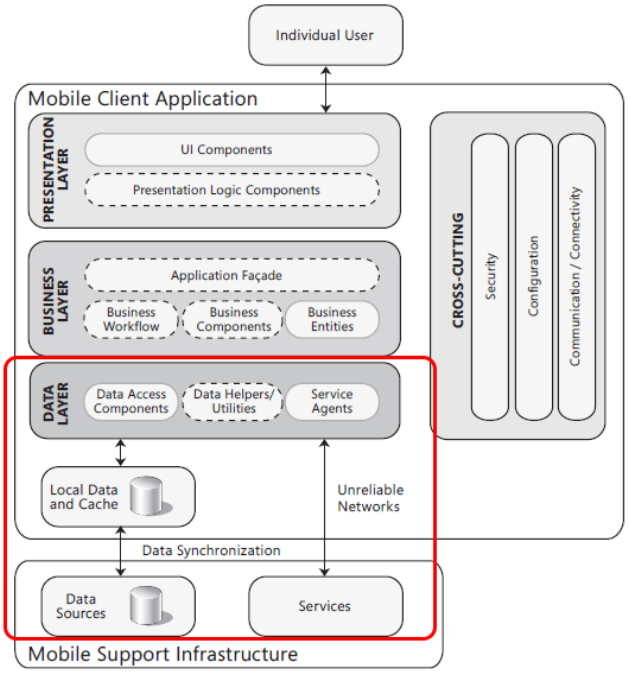


Figura 3.3 Capa de datos de una aplicación móvil. *Fuente: Microsoft Application Architecture Guide [46]*

En el diseño arquitectural que nos presenta MAAG, la capa de datos se encarga de los datos del sistema (datos locales y caché, *Local Data and Cache*), así como también de los datos provenientes de otros sistemas (por ejemplo, base de datos, archivos compartidos, otros servicios y cualquier otra aplicación que administre los datos), llamados Fuentes de Datos (o *Data Sources*) [46]. Ribeiro y Silva [51] llaman a estas fuentes: proveedores (en inglés, *providers*). En este PFC seguiremos esta denominación, referiéndonos a las fuentes de datos como proveedores de datos.

La propuesta que presentamos está basada en este modelo arquitectural, abarcando el diseño de la capa de datos de las aplicaciones móviles, donde consideramos los dos aspectos principales identificados en esta capa: persistencia (almacenamiento local) y proveedores de datos (ver capítulo 5).

3.4 Síntesis del capítulo

El problema de naturaleza transitoria de los teléfonos móviles, implica tomar medidas a aquellas aplicaciones que están en constante conexión a Internet realizando peticiones a servicios web. La persistencia de datos o el almacenamiento local, es una necesidad en caso que no se cuente con esta conexión.

Existen opciones comunes de almacenamiento en la actualidad, como HTML5 (para aplicaciones híbridas), el almacenamiento en servicios de alojamiento en la nube (como Google Drive y Amazon S3) y las bases de datos. Categorizando estas opciones encontramos que el almacenamiento puede ser local o remoto. Por una parte, entre los mecanismos de almacenamiento local más comunes en los teléfonos móviles encontramos los pares clave-valor, archivos, base de datos integradas y almacenamiento en dispositivos externos (por ejemplo, tarjetas de memoria SD). Por otra parte, el almacenamiento remoto implica el almacenamiento en la nube (servicios de alojamiento, servidores y base datos externos).

Son varios los mecanismos de persistencia local encontrados, y es importante destacar que todas estas tienen soporte en los diferentes sistemas operativos móviles, pero cada una lo maneja siguiendo diferentes caminos, notándose así que el problema de la fragmentación también provoca dificultades al momento de trabajar con el diseño de la persistencia para una aplicación móvil.

Siguiendo la arquitectura multicapas, la capa de datos se encarga de cubrir los aspectos referentes a la persistencia en las aplicaciones móviles. De la misma forma, se obtienen datos de proveedores, los cuales luego son almacenados. Los datos involucrados pueden ser de distintos tipos: imágenes, documentos, configuraciones del usuario, entre otros. Dependiendo si los datos deben almacenarse persistentemente o temporal, existen mecanismos de persistencia que ayudan a llevarlo a cabo.

Capítulo 4

Desarrollo Móvil Dirigido por Modelos

La Ingeniería de Software Dirigida por Modelos, del inglés *Model Driven Software Engineering* (MDSE) puede ser definida como una metodología en la cual se aplican las ventajas del modelado en las actividades de ingeniería de software. MDD, a su vez, se considera como un subconjunto de MDSE [11]. En este capítulo se analiza este enfoque de desarrollo para la generación de aplicaciones móviles.

En la sección 4.1 se brinda los conceptos básicos de este enfoque de desarrollo, y se habla particularmente de dos conceptos relacionados a MDD: MDA y Modelado Específico del Dominio, del inglés *Domain Specific Modeling* (DSM).

Luego, en la sección 4.2 se analiza este enfoque aplicado al desarrollo móvil. Se habla de las propuestas MDD para el desarrollo de aplicaciones móviles, y se discuten algunos aspectos resaltantes encontrados.

Por último, en la sección 4.3 se presenta a MoWebA, un enfoque orientado al desarrollo web, pero que consideramos podría ser adecuado para el desarrollo de aplicaciones móviles teniendo en cuenta ciertas características que podrían tener un impacto positivo en el desarrollo estas aplicaciones. Este enfoque constituye la base de nuestra propuesta para el desarrollo de aplicaciones móviles.

4.1 Desarrollo Dirigido por Modelos (MDD)

MDD es un paradigma de desarrollo de software cada vez más aceptado y usado por la comunidad. Promete mejorar el proceso de construcción de software basándose en un proceso guiado por modelos y soportado por potentes herramientas; además, asigna a los modelos un protagonismo que los hace tan importantes como el código fuente [47]. Este paradigma de desarrollo permite mejorar las prácticas comunes de desarrollo de software, presentando ventajas tales como [56]:

- **incremento en la productividad**: reduciendo los costos de desarrollo gracias a la generación automática del código y otros artefactos a partir del modelo, incrementando la productividad de los desarrolladores;
- **adaptación a los cambios tecnológicos**: los modelos de alto nivel están libres de detalles de la implementación, lo que facilita la adaptación a los cambios que pueda sufrir la plataforma tecnológica subyacente o la arquitectura de la implementación;
- **adaptación a los cambios en los requisitos**: con toda la información capturada en las transformaciones para generar los artefactos de implementación, el adaptarse a un cambio o adición de un requisito utilizando MDD, es una tarea sencilla mediante la reutilización de dicha información;
- **consistencia**: mediante la automatización, antes que la generación manual, MDD favorece la consistencia de los artefactos generados;
- **reutilización**: en MDD la inversión está en el desarrollo de modelos y transformaciones. A medida que éstos se van reutilizando, se gana en productividad y en adaptación a cambios de requisitos.

Dos enfoques muy relacionados con los conceptos de MDD son **MDA** y **DSM**. El primero tiende a enfocarse en modelado bajo los estándares y guías establecidos por el Grupo de Gestión de Objetos, del inglés *Object Management Group* (OMG)[1]. El segundo, sin seguir un estándar específico, permite modelar para un dominio específico.

Arquitectura Dirigida por Modelos (MDA). Con MDA se define una arquitectura que proporciona un conjunto de guías para estructurar especificaciones expresadas como modelos, siguiendo el proceso MDD [47] [56]. Acorde con las directivas de la OMG, las dos motivaciones principales para MDA son la *interoperabilidad* (independencia de los fabricantes a través de la estandarización) y la *portabilidad* (independencia de plataforma) de los sistemas de software. Así mismo, promete mejorar la *mantenibilidad* de los sistemas a través de la separación de conceptos [56]. De esta manera, MDA propone una representación independiente de los aspectos del sistema mediante [20] [47]:

- Modelo Independiente de la Computación, del inglés *Computation Independent Model* (CIM), también conocido como modelo del dominio, el cual no muestra detalles de la estructura del sistema;

[1] OMG. Es un consorcio dedicado a la gestión y el establecimiento de diversos estándares de tecnologías orientadas a objetos, tales como UML, XMI, CORBA. Es una organización no lucrativa que promueve el uso de tecnología orientada a objetos mediante guías y documentos de especificación de estándares. Enlace: http://www.omg.org

- Modelo Independiente de la Plataforma, del inglés *Platform Independent Model* (PIM), es una vista con un alto nivel de abstracción, donde las funcionalidades de negocios y comportamiento están separadas de cualquier tecnología o lenguaje de implementación, lo que permite ser implementado sobre distintas plataformas específicas;
- PSM, representa la proyección del PIM en una plataforma específica; y, el
- Modelo de la Implementación (Código), constituye el último paso en el desarrollo: la transformación del PSM a código fuente .

Las especificaciones de MDA[2] establecen que el desarrollo de software en MDA empieza con el PIM de una aplicación (funcionalidades de negocio y comportamiento), elaborado usando un lenguaje de modelado basado en Meta-Object Facility (MOF)[3] de la OMG. Este modelo permanece estable a medida que la tecnología evoluciona. Las herramientas de desarrollo de MDA[4] convierten primero el PIM a PSM, y luego a una implementación funcional en prácticamente cualquier plataforma *middleware*: servicios web, XML/SOAP, EJB, C#/.NET, entre otros. La portabilidad y la interoperabilidad están integradas en la arquitectura. Las especificaciones de modelado estándar de la industria de la OMG apoyan MDA, tales como el Unified Modeling Language (UML)[5], MOF, XML de Intercambio de Metadatos (XMI)[6] y el Common Warehouse Metamodel (CWM)[7] [47].

Modelado Específico del Dominio (DSM). DSM propone la idea de crear modelos específicos para un dominio o un área de interés de desarrollo en particular, mediante la utilización de lenguajes especializados para dicho dominio. Estos lenguajes se conocen como DSLs y utilizan directamente conceptos del dominio del problema para especificar una solución [47] [49]. Los DSL son utilizados para construir lenguajes de modelado, por lo general son gráficos. La principal diferencia del uso de modelos respecto a MDA, es que éstos no utilizan ningún estándar de la OMG para su infraestructura (no están basados en UML) [47].

Para trabajar con DSM, encontramos una familia importante de herramientas denominada *"Software Factories"*. Este término, introducido por Microsoft, hace referencia a una línea de producción de software que configura herramientas de desarrollo extensibles, como por ejemplo *Visual Studio Team Services*[8], con contenido empaquetado como DSLs, *frameworks*

[2] *MDA® Specifications*. http://www.omg.org/mda/specs.htm#MDAapps
[3] MOF. http://www.omg.org/mof/
[4] MDA, *Committed Companies And Their Products*. http://www.omg.org/mda/committed-products.htm
[5] UML. http://www.uml.org/
[6] XMI. http://www.omg.org/spec/XMI/
[7] CWM. http://www.omg.org/spec/CWM/
[8] *DevOps overview for Team Services and TFS*. https://www.visualstudio.com/es-es/docs/devops-alm-overview

y herramientas para generar tipos específicos de aplicaciones para diferentes plataformas. Este tipo de herramientas asisten en la tarea de construcción de modelos y lenguajes de modelado [47].

4.2 Desarrollo móvil dirigido por modelos

La heterogeneidad en las plataformas móviles repercute en el esfuerzo de desarrollo de las aplicaciones móviles para cada plataforma [11] [5]. Desarrollar aplicaciones que comparten las mismas funcionalidades y comportamiento, pero que son destinados a diferentes plataformas, es un área adecuado para MDD [23]. Con MDD, los desarrolladores de aplicaciones, describen sus aplicaciones en un nivel alto de abstracción mediante modelos independientes de detalles específicos de la plataforma, tanto para la presentación, la lógica de negocios y el acceso de datos. Esta especificación es traducida a código para diferentes plataformas. Esto, siguiendo el enfoque MDA, significa implementar PIM para los diversos aspectos de los móviles (modelos independientes de aspectos tecnológicos), y luego mediante transformaciones sucesivas, llegar al PSM para cada plataforma a la que se desea implementar la aplicación [56] [5] [49] [47]. Así, no tenemos solamente un código base multiplataforma (el modelo), sino también un incremento en el nivel de abstracción y ciclos rápidos de desarrollo.

Del estudio llevado a cabo en la sección 2.3, pudimos obtener unos primeros juicios sobre las propuestas MDD disponibles para el desarrollo de aplicaciones móviles. A continuación presentamos un estudio más completo y detallado llevado a cabo. Veremos cuáles son algunas de las propuestas disponibles y realizaremos un análisis comparativo entre ellas.

4.2.1 Propuestas MDD para el desarrollo de aplicaciones móviles

El principal objetivo del estudio realizado fue obtener una visión global de MDD en el marco del desarrollo de las aplicaciones móviles. Nos enfocamos en el desarrollo de aplicaciones móviles nativas, por ser éstas las principales víctimas de la fragmentación (sección 2.3). Por otra parte, en la sección 3.2.2 mencionamos que los mecanismos de persistencia se manejan siguiendo caminos diferentes en cada sistema operativo móvil, destacando la dificultad de su uso ocasionado por el problema de la fragmentación. A partir de esto, guiamos nuestro estudio enfocados a analizar la persistencia desde el punto de vista de las propuestas MDD.

Con el fin de analizar las propuestas MDD para el desarrollo de aplicaciones móviles, llevamos a cabo un estudio de la literatura disponible siguiendo algunas de las pautas

propuestas para realizar mapeos sistemáticos de la literatura [31], sin seguir estrictamente ese método:

- La búsqueda fue realizada en bibliotecas digitales como *IEEE Xplore*, *ACM*[9], *SpringerLink* y *ResearchGate*[10]. De la misma forma, utilizamos el buscador de Google destinado a la búsqueda de literatura científica-académica, *Google Scholar*.
- Nuestras cadenas de búsqueda estuvieron basadas en aspectos de nuestro interés: MDD, móviles y persistencia.
- Los criterios de exclusión aplicados a los trabajos revisados fueron: i) no tienen nada que ver con el proceso MDD y/o aplicado a móviles; ii) año de publicación a partir del 2005.

Como resultado del estudio realizado, los cuadros A.1 y A.2 del Anexo A muestran las propuestas recopiladas para el estudio: 18 soluciones MDD para el desarrollo de aplicaciones móviles y sus respectivos aportes. Los cuadros 4.1 y 4.2 comparan cada propuesta recopilada, realizando una descripción breve, en base a diversos aspectos que nos parecen interesantes:

A) Análisis de enfoque MDD utilizado por cada propuesta, recopilamos los siguientes datos:

   A.1) qué aspectos de la aplicación móvil fueron tenidos en cuenta por cada propuesta en su desarrollo (GUI, lógica de negocios, datos, entre otros);

   A.2) en qué se basaron para el modelado (por ejemplo, en perfiles UML, en un DSL, entre otros);

   A.3) qué lenguaje de modelado fue utilizado (por ejemplo, UML, DSL, y otros);

   A.4) qué tipo de lenguaje modelado representa (gráfico, textual); y por último,

   A.5) si se basaron en estándares de la OMG (MDA, UML y IFML).

B) Aspectos del desarrollo para móviles fueron analizados mediante:

   B.1) cuáles fueron las plataformas de destino elegidas por cada herramienta y/o propuesta (por ejemplo, Android, iOS, entre otros);

   B.2) si se utilizó MVC como patrón de arquitectura;

   B.3) si generaban aplicaciones nativas;

[9] ACM. http://dl.acm.org/dl.cfm
[10] ResearchGate. https://www.researchgate.net/

B.4) si estas aplicaciones generadas eran completas[11];

B.5) de qué tipo eran las aplicaciones generadas (por ejemplo, juegos, aplicaciones de negocio, entre otros).

C) A partir de las propuestas estudiadas y analizando la utilización de la persistencia en ellas (aspectos tenidos en cuenta para cubrir la persistencia), mencionamos aspectos comunes encontrados, los cuales creemos nos permitirán tener un mejor panorama de la utilización de la persistencia en el desarrollo de aplicaciones móviles. Tenemos así:

C.1) si se consideraron aspectos de persistencia en el modelado;

C.2) si soporta la modalidad "sin conexión";

C.3) si contempla la conexión con alguna base de datos externa a la aplicación.

D) Como analizamos propuestas de investigación al igual que nuestro PFC, nos pareció interesante averiguar:

D.1) los métodos de evaluación utilizados, y los resultados obtenidos en el desarrollo de sus propuestas.

[11] Consideramos como completa una aplicación que al ser generada es totalmente funcional, es decir, en su desarrollo se tuvo en cuenta más aspectos que solamente la GUI, como por ejemplo lógica de negocios y datos.

Propuestas basadas en MDD para el desarrollo de aplicaciones móviles

Propuesta	Aspectos considerados	En qué se basa para el modelado	Lenguaje de modelado	Tipo de lenguaje de modelado	Utiliza estándar OMG	Plataformas posibles de destino	Utiliza MVC
P1	Estructural y lógica de negocios	Perfil UML	UML	Gráfico	UML	Android y Windows Phone[12]	No
P2 y P3	GUI	DSL	UML, Technology Neutral DSL	Textual	MDA	Android[12]	No
P4	GUI y lógica de negocios	EMF	EMF	Gráfico	No	Android y iOS	Sí
P5	GUI y lógica de negocios	IFML	IFML para móvil	Gráfico	IFML	Android y iOS[12]	No
P6	Estructural, lógica de negocios y GUI	IFML	IFML para móvil (interfaz) y Flujo Visual (*backend*)	Gráfico	IFML	Android y iOS	No
P7	No especificado	λ calculus	MobDSL	Textual	No	Android y iOS	No especificado
P8	Estructural, lógica de negocios y GUI	Perfil UML llamado XIS	XIS-Mobile DSL	Gráfico	No	Android, iOS y Windows Phone[12]	No
P9	Datos, estructural, lógica de negocios, y GUI	DSL	RAPPT DSL	Gráfico/Textual[13]	No	Android	Sí
P10	Estructural, lógica de negocios y GUI	Perfil UML	UML	Gráfico	MDA	Android y Windows Phone	Sí
P11	GUI	MIM	MIM DSL	Gráfico	No	No especificado	No
P12	Requerimientos y lógica de negocios, y GUI	DSL	AXIOM DSL	Textual	MDA	Android y iOS[12]	Sí
P13	Datos, GUI y lógica de negocios	DSL	Xmob DSL(PIM); perfil UML(PSM)	Textual	MDA	Android, iOS y .NET	Sí
P14 y P15	Datos, GUI y lógica de negocios	DSL	MD2 DSL	Textual	No	Android y iOS	Sí
P16	GUI	UML	UML	Gráfico	MDA	Android y Blackberry	No
P17	Datos, GUI, lógica de negocios y recursos de hardware	Perfil UML	UML	Gráfico	UML	Windows Phone 7	Sí
P18	Datos, GUI, lógica de negocios y recursos de hardware	Perfil UML	UML	Gráfico	UML	Android	Sí

Cuadro 4.1 Propuestas MDD resultantes del estudio de la literatura (Parte I)

[12]El lenguaje soporta cualquier plataforma, pero la herramienta desarrollada soporta solo estas plataformas.

[13]Gráfico, modelado de la navegación de la aplicación. Textual, para la descripción y diseño de cada pantalla.

[14]Se nombra un estereotipo *"Persistence"* en el perfil UML. Se usa un campo *"Persistencetype"* para indicar el tipo de persistencia. Se usa como ej. para modelar la clase *DBHandler* o manejador de la base de datos.

[15]Basadas en HTML5, CSS y JavaScript. Optimizado para el framework Apache Cordova.

[16]El caso de estudio contempla el desarrollo de una aplicación con soporte para bases de datos SQLite.

[17]Las entidades poseen un campo boolean *"Persistent"*, que indica si deben o no ser persistentes.

[18]Generación parcial de la aplicación, solo la GUI.

[19]XMOB-data tiene en cuenta el mecanismo de datos a utilizar, especificamente si se trata o no de consumir datos de un servicio. Se especifica el *bean* y *datasource*, y el tipo de dato de retorno.

[20]Genera totalmente la aplicación, sin necesidad de compilar el código generado en un IDE.

[21]Utiliza el *contentProvider* / ORM para acceder a los datos. Se realizan acciones CRUD para persistir datos. Se especifica con *"providerType"* el tipo de almacenado, ya sea local (archivos) o remoto (*backend*).

[22]En el metamodelo se contempla el *"Isolate Storage"*, espacio de almacenamiento Windows Phone.

[23]En el metamodelo se tiene en cuenta el *ContentProvider*, propio de Android,

Propuestas basadas en MDD para el desarrollo de aplicaciones móviles

Propuesta	Generación de código nativo	Generación completa de la aplicación	Tipo de aplicación generada	Presenta evaluación de su propuesta	Sus evaluaciones dieron resultados positivos	Tiene en cuenta la persistencia en el modelado	Soporte modalidad "sin conexión"	Permite conexión con bases de datos
P1	Sí	Sí	Soporta el desarrollo de todo tipo de aplicaciones	Sí	Sí	Sí[14]	No especificado	No especificado
P2 y P3	Sí	No	No especificado	Sí	Sí	No aplica	No aplica	No aplica
P4	Sí	Sí	Aplicaciones de negocio dirigidas por datos	Sí	Sí	No especificado	Sí	No especificado
P5	No, *native wrapper*	Sí	Aplicaciones móviles basadas en tecnologías web[15]	Sí	Sí	No aplica	No aplica	No especificado
P6	No, *native wrapper*	Sí	Aplicaciones móviles basadas en tecnologías web[15]	No	No aplica	No especificado	Sí	Sí
P7	No, interpreta el código a través de la máquina virtual	Sí	Aplicaciones dirigidas por datos	Sí	Sí	No especificado	No especificado	Sí[16]
P8	Sí	Sí	Aplicaciones dirigidas por datos	Sí	Sí	Sí[17]	No especificado	No especificado
P9	Sí	Sí	Aplicaciones dirigidas por datos	Sí	Sí	No especificado	No especificado	Sí
P10	Sí	Sí	Aplicaciones sencillas, que no requieran gráficos avanzados	Sí	Sí	No especificado	No especificado	No especificado
P11	No especificado	No[18]	Aplicaciones dirigidas por datos	Sí	Sí	No aplica	No especificado	Sí
P12	Sí	Sí	Soporta el desarrollo de todo tipo de aplicaciones	Sí	Sí	No especificado	No especificado	No especificado
P13	Sí	Sí	Aplicaciones dirigidas por datos	No	No aplica	Sí[19]	No especificado	Sí
P14 y P15	Sí	Sí[20]	Aplicaciones dirigidas por datos	Sí	Sí	Sí[21]	No especificado	No especificado
P16	No especificado	No[18]	No especificado	Sí	Sí	No aplica	No aplica	No aplica
P17	No especificado	No especificado	No aplica	Sí	Sí	Sí[22]	No especificado	No especificado
P18	No especificado	No especificado	No aplica	Sí	Sí	Sí[23]	No especificado	No especificado

Cuadro 4.2 Propuestas MDD resultantes del estudio de la literatura (Parte II)

4.2.2 Discusión sobre resultados

Teniendo en cuenta los criterios mencionados, a continuación analizaremos los resultados más relevantes obtenidos.

A.5) Entre todas las propuestas con enfoque dirigido por modelos, tenemos que 6/18 propuestas (33%) adoptan o se basan en estándares MDA, utilizando separación de conceptos y diseño mediante PIM y PSM; todas éstas utilizan UML como lenguaje de modelado, a excepción de P12, que utiliza lenguaje textual llamado AXIOM DSL. También, encontramos que P1, P17 y P18 utilizan UML como estándar. Por otra parte, P5 y P6 utilizan el estándar IFML de la OMG para interfaces de sistemas, incluido móviles.

B.1) El sistema operativo más utilizada por las propuestas es Android, utilizada por la mayoría para ser destino de ejemplos, casos de estudio y de evaluaciones. Como el desarrollo es multiplataforma, muchos casos de estudio generan a su vez para iOS, específicamente encontramos 7/18 propuestas (39%) que generan para Android y iOS. Cabe resaltar que Windows Phone es también una plataforma muy utilizada, apareciendo en 4/18 propuestas (22%) como plataforma de prueba.

B.3) Casi todas las aplicaciones generadas son nativas, por lo que se puede ver una tendencia hacia este tipo de aplicación. En total tenemos a 9/18 propuestas (50%) que son nativas, 3/18 (17%) serían híbridas, y los restantes, no especificaron.

B.4) Es muy importante notar las estrategias de generación que encontramos. Tenemos que 11/18 propuestas (61%) sí generan aplicaciones completas. Entre estas, solamente P14 y P15 generan la aplicación en su totalidad, es decir, generan las estructuras del proyecto y la aplicación final, sin necesidad de una compilación previa en un IDE, como las demás. Esta propuesta es MD2, y afirma que el trabajo más costoso fue la generación para iOS, puesto que debían generar incluso archivos temporales para emular lo que la herramienta Xcode hace para poder compilar la aplicación.

B.5) Es importante aclarar algo que todas las propuestas decían: los DSL desarrollados tienen la capacidad y soporte para generar todo tipo de aplicaciones; pero, la complejidad de la aplicación es proporcional a la dificultad y el esfuerzo en el modelado y en la generación de código. Para la mayoría de los casos de estudio y evaluaciones, se optaron por aplicaciones que no requerían muchos gráficos ni mucha interacción con el usuario. Es así que tenemos que las aplicaciones orientadas por datos fueron las más elegidas, por 6/18 propuestas (33%).

C.1) Este punto nos brinda una visión del grado de detalle y consideración que tuvieron las propuestas respecto a la persistencia. Anteriormente hablamos de la importancia de tener en cuenta este aspecto en las aplicaciones móviles. Nos interesa saber qué tipo de técnicas u opciones de almacenamiento fueron utilizadas por las distintas propuestas.

- Encontramos que 6/18 propuestas (33%) no especifican, según nuestro estudio, ni hacen mención o consideran la persistencia al momento de modelar; a su vez, 5/18 propuestas (28%) no lo aplican, puesto que abarcan solamente aspectos de interfaz. Pero encontramos 7/18 propuestas (39%) que de alguna u otra forma sí tienen en cuenta en el modelado algún aspecto de persistencia.

  Resumiendo algunas formas de persistencia encontradas: P1 en su perfil UML móvil nombra un estereotipo *"Persistence"*, junto con un atributo *"Persistencetype"* para indicar el tipo de persistencia (este atributo es clave puesto que puede contemplar distintos tipos de persistencia); pero, no se brindan mayores detalles ni se cuenta con una especificación del perfil utilizado (no habla de cuáles son esos tipos, ni hasta qué punto los tiene en cuenta). Por otra parte, P17 presenta en su perfil UML para Windows Phone el estereotipo *"IsolateStorage"*, tratándose del almacenamiento específico de Windows Phone; pero, no encontramos más información al respecto. Lo mismo pasa con P18, donde se tiene en cuenta el *ContentProvider* (específico de Android), indicando la forma en que se almacenan los datos, teniendo como posibles valores: *SharedPreferences*, *InternalStorage*, entre otros. P8, a su vez, hace uso de un campo *"Persistent"* en sus entidades para indicar que esa entidad debe ser persistente. Resulta muy interesante, como método de selección, qué campos o qué datos deben ser almacenados. En P13 encontramos un modelo dedicado a datos, donde se detalla la conexión con una fuente de datos (o *datasource*) local (se especifica el tipo de datos que tiene de entrada y de salida) o remoto (se especifica la URL de conexión). De la misma forma, encontramos que P14 y P15 especifican con un *"providerType"* este tipo de detalle de la fuente de datos, indicando que el almacenamiento local lo realiza por medio de archivos y el remoto, mediante conexión a un *backend*.

C.2) Una conexión con una base de datos externa implica el intercambio de datos entre el dispositivo y una base de datos, lo que implicaría la necesidad de guardar datos localmente (el uso de la técnica de guardar datos en caché podría ser interesante como opción en caso que no haya conexión).

- Tenemos 5/18 propuestas (28%) que contemplan la conexión con una base de datos externa, incluyendo en el modelo las configuraciones de conexión con la misma. Pero sin dar detalle de qué hace con la información recibida.

C.3) La modalidad "sin conexión" puede realizarse con algún mecanismo que permita persistir datos en el teléfono (como base de datos y archivos), a fin de poder utilizarlos en caso que no haya conexión de red.

- Considerando las propuestas del punto ii), solamente P6 permite el soporte para la modalidad "sin conexión" y permite conexión externa a una base de datos. P6 sería la opción más robusta puesto que asegura que aún se podría contar con datos en el teléfono cuando no haya conexión, pero no se brindan más detalles al respecto. No pasa así con P7, P9, P11 y P13, los cuales no especifican si soportan ésta modalidad. Por último, P4 soporta la modalidad sin conexión, pero no especifica si permite o no conexión con bases de datos externas.

En general fue muy poco lo que pudimos encontrar. El manejo de la persistencia es un tema no muy claro ni muy especificado entre las propuestas analizadas. Encontramos indicios de algunas formas de implementación de la persistencia, analizando los metamodelos y perfiles (si los presentaban), como por ejemplo indicar si el almacenamiento es local o remoto, o si una entidad o clase debía ser persistente o no. Pero somos conscientes de que capaz muchas propuestas abarcaban aspectos de los móviles a nivel PIM, por lo que la ausencia de detalles de persistencia a ese nivel tendría sentido, pero en base a lo poco recopilado, no podemos decir más.

Destacamos que P1 es el único que menciona la persistencia y da a entender que podría manejar distintos tipos de persistencia (por los estereotipos que encontramos en su perfil). Pero este no indica cuáles podrían ser estos tipos o mecanismos de persistencia, ni brinda más detalles al respecto. En este sentido encontramos una oportunidad de investigación: *cubrir conceptualmente la persistencia en el desarrollo de aplicaciones móviles, mediante elementos específicos que nos permitan tener en cuenta las distintas opciones y mecanismos de persistencia en el modelado, a fin de permitir el almacenamiento de datos incluso sin conexión de red, y considerando la conexión con fuentes remotas*.

D.1) En general todos los trabajos proponen un método de evaluación para sus propuestas, y todas con resultados positivos; solamente P6 y P13 no dan más detalles de la evaluación utilizada. Entre los métodos de evaluación utilizados tenemos:

- Implementación de una misma aplicación en varias plataformas, verificando así la generación multiplataforma del modelo desarrollado; es el caso de P2, P10, P12 y P16.
- Otro muy utilizado es simplemente un caso de estudio demostrando la aplicabilidad de la propuesta en una aplicación; 11/18 propuestas (61%) lo llevaron a cabo. Entre estas, P1 y P10 compararon la aplicación generada con una ya existente.

- Varias propuestas hicieron experimento con un grupo de personas. P5 trabajo con un grupo de desarrolladores, haciendo comparación de esfuerzo. P9 hizo lo mismo, pero fueron desarrolladores que probaron la herramienta quienes dieron sus evaluaciones positivas sobre ella. P11 llevó a cabo un estudio de aplicabilidad del DSL desarrollado, donde varios grupos de trabajo desarrollaron una aplicación de prueba, luego respondieron ciertas preguntas. Al final, la hipótesis de adopción del lenguaje fue aceptada tras los comentarios positivos de los usuarios.
- Resalta que P15 lleva a cabo una comparación por líneas de código de la aplicación generada. No nos parece una buena métrica de comparación, puesto que el esfuerzo es menor porque se generó código automáticamente, no porque haya menos líneas. Además, en una aplicación desarrollada tradicionalmente, la cantidad de líneas varía según la habilidad del desarrollador.
- Las métricas más utilizadas fueron: comparación de esfuerzo (utilizando o no la propuesta), comparación de la aplicación generada con una ya existente, generación multiplataforma, y experimento con un grupo de gente (tanto desarrolladores como gente ajena al área) utilizando la herramienta y recibiendo comentarios.

4.2.3 Puntos resaltantes obtenidos

- Ninguna de las propuestas adopta todos los aspectos considerados de la persistencia. Ante la necesidad de especificaciones para describir la persistencia en el modelado de aplicaciones móviles, pretendemos cubrir conceptualmente la persistencia en el desarrollo de aplicaciones móviles, mediante elementos específicos que nos permitan tener en cuenta las distintas opciones y mecanismos de persistencia en el modelado, a fin de permitir el almacenamiento de datos incluso sin conexión de red, y considerando la conexión con fuentes remotas.
- Poca adopción de MDA (en especial del estándar MOF).
- Android y iOS constituyen las plataformas destino más elegidas para los casos de estudio y evaluaciones. Como tercera opción se encuentra Windows Phone.
- Las aplicaciones generadas son en gran mayoría nativas y orientadas a datos.
- La mayoría de las propuestas generan la aplicación teniendo en cuenta la estructura del proyecto según el IDE correspondiente a la plataforma elegida; mediante esto se puede realizar una compilación con el SDK respectivo y así obtener la aplicación ejecutable.

- El tipo de evaluación más visto es la de realizar una ilustración o caso de estudio de la propuesta presentada.

Del estudio destacamos la necesidad de especificaciones para describir la persistencia en el modelado de aplicaciones móviles y la poca adopción de MDA (en especial del estándar MOF). Se pretende cubrir conceptualmente la persistencia en el desarrollo de aplicaciones móviles, mediante elementos específicos que permitan tener en cuenta las distintas opciones y mecanismos de persistencia en el modelado, a fin de posibilitar el almacenamiento de datos incluso sin conexión de red, y considerando la conexión con fuentes remotas.

4.3 MoWebA

MoWebA se presenta como una metodología MDD enfocada al desarrollo de aplicaciones web centrado en roles. Posicionada dentro del marco investigativo del proyecto MDD+, esta propuesta adopta los estándares MDA en cada fase (lenguajes, herramientas, arquitectura, entre otros) y presenta una arquitectura por capas. La separación de conceptos (contenido, navegación, procesos y presentación) ya identificados en otras propuestas, es considerada fuertemente y reforzada, añadiendo a los usuarios como un nuevo concepto de separación [20].

Para formalizar los procesos de modelado y transformación, MoWebA adopta el lenguaje MOF para la definición de la sintaxis abstracta, y la extensión del perfil UML para una definición precisa del lenguaje de modelado. Además, es posible llevar a cabo extensiones a sus metamodelos para cubrir nuevas características, lo cual la hace adaptable a los cambios actuales [20].

La figura 4.1 presenta las dimensiones de MoWebA, que cubre los procesos de modelado y transformación. Para el modelado, MoWebA adopta el enfoque MDA identificando tres abstracciones diferentes [20]:

- el espacio del problema, cubierto por los modelos CIM y PIM;
- el espacio de la solución, el cual es cubierto por el ASM y PSM;
- definición de código fuente, cubierto por el modelo Modelo Específico de la Implementación, del inglés *Implementation Specific Model* (ISM) y ajustes manuales

4.3.1 Proceso de modelado y transformación

MoWebA define dos procesos principales complementarios: el primero, relacionado a las actividades de modelado; y el segundo, a las actividades de transformaciones.

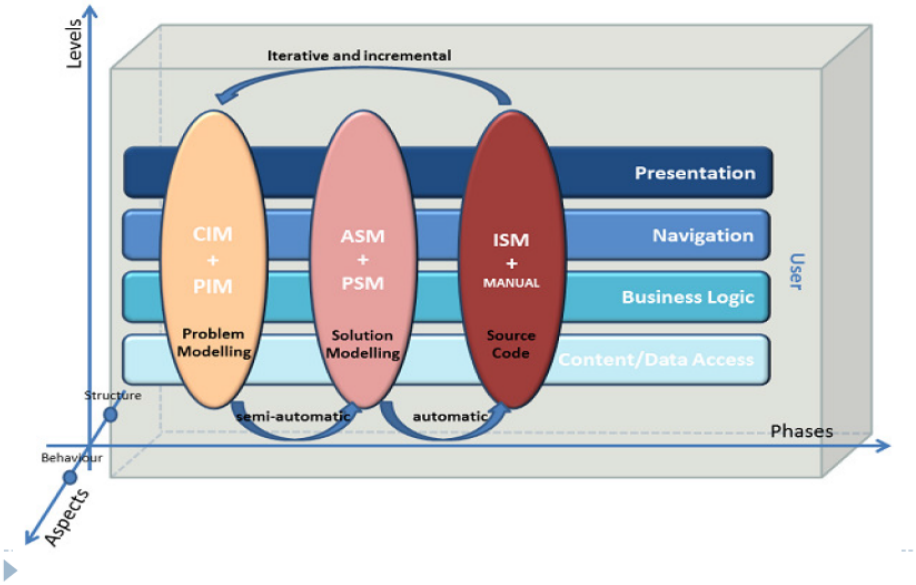


Figura 4.1 Dimensiones de MoWebA

Proceso de modelado. Consiste en la utilización y la especificación sistematizada de los modelos CIM, PIM y PSM descritas por MDA, y del modelo específico propuesto por MoWebA, ASM. Como se ve en la figura 4.2 se establecen siete etapas: etapas del 1 al 6, donde se definen el CIM y PIM; y la etapa 7 donde se definen el ASM y PSM [20]. A continuación haremos una breve descripción de cada una de las etapas.

En la **etapa 1** se realiza el análisis de requerimientos. Aquí los potenciales usuarios son identificados. Por cada potencial usuario se lista una serie de requerimientos funcionales, navegacionales y de usabilidad. Constituye el punto de partida para la definición del Árbol Navegacional. El diagrama producido en esta etapa es el Diagrama de Casos de Uso. La **etapa 2** significa la definición del Diagrama de Entidades, donde se organiza la información estructuralmente. Aquí se define el Árbol Navegacional. Los diagramas de Rol y Zonas son utilizados, considerando los usuarios potenciales de la etapa 1. Los diagramas producidos en esta etapa son el de Entidades, el Árbol Navegacional y los diagramas de Rol y Zona. En la **etapa 3** se define el comportamiento de cada nodo del árbol navegacional a través del Diagrama de Nodos (representado por Diagrama de Estados de UML). La **etapa 4** se encarga de la presentación, donde se definen qué elementos van a ser mostrados en cada página mediante el Diagrama de Contenido. En esta etapa, las estructuras de las páginas (posiciones de la cabecera, menús, pie de página, entre otros) son definidas a través del Diagrama de Estructuras. La composición estructural de los procesos de negocios y los procedimientos transaccionales también son definidos es ésta etapa con el Diagrama Lógico. De esta forma,

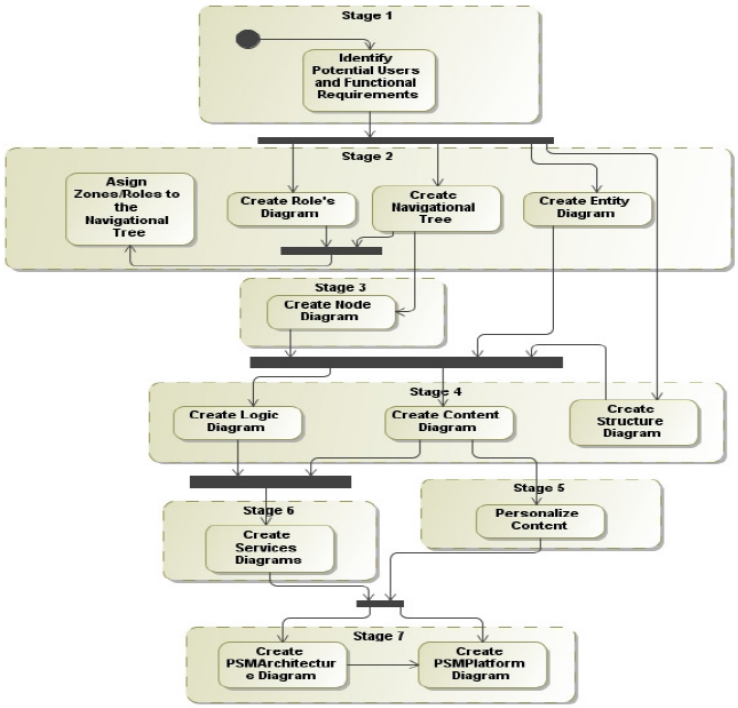


Figura 4.2 Etapas de modelado de MoWebA

los diagramas producidos son el de Contenido, de Estructura y el Lógico. Para la **etapa 5**, la personalización de los modelos es realizada a través del Modelo de Adaptación. En esta etapa, MoWebA propone definir los Diagramas de Fuente y Reglas. En la **etapa 6** se propone una definición detallada de cada servicio o acción identificada en el Diagrama Lógico y de Contenido usando el Diagrama de Servicios. Por último, en la **etapa 7** se generan los diagramas ASM y PSM. Aquí se propone un enriquecimiento de los modelos existentes en orden a considerar aspectos relacionados a la arquitectura final del sistema y agregando información específica de la plataforma [20].

Proceso de definición del ASM. ASM incorpora conceptos, patrones y funciones procedentes de arquitecturas emergentes (por ejemplo, RIAs, arquitectura móvil). Como vemos en la figura 4.3, para definir el ASM, MoWebA propone los siguientes pasos:

1) Definir el metamodelo móvil usando MOF.

2) Definir el correspondiente perfil UML móvil.

3) Definir las reglas de transformación para los elementos que serán obtenidos de manera automática. Si el ASM ya existe:

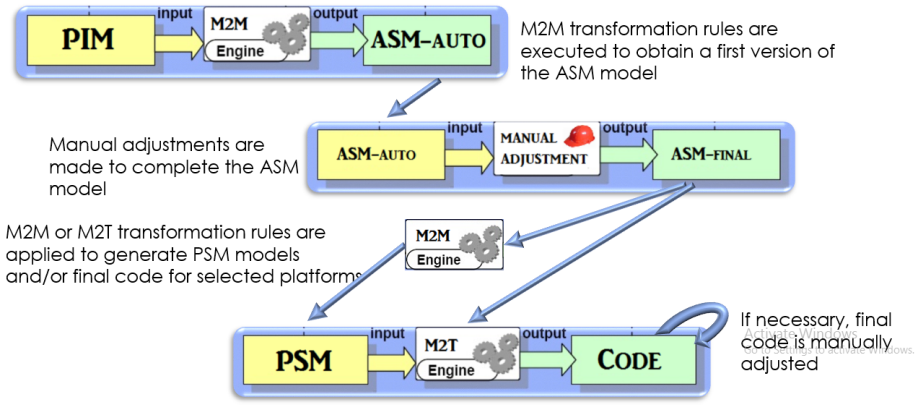


Figura 4.3 Proceso de definición del ASM

4) Aplicar reglas de transformación a fin de obtener la primera versión del modelo ASM.

5) Realizar los ajustes manuales necesarios para completar el modelo ASM.

6) Generar los modelos PSM para las plataformas seleccionadas, o el código final aplicando las reglas de transformación.

7) Si es necesario, realizar ajustas manuales.

Como podemos ver en el proceso previo, como consecuencia de ofrecer una mayor reusabilidad del PIM y facilitar la evolución arquitectural de las aplicaciones web, el enfoque MoWebA requiere de algunos esfuerzos adicionales, incluido la necesidad de especificar los metamodelos ASM y la definición de las correspondientes reglas de transformación, a fin de lograr transformaciones automáticas en la arquitectura y plataforma propuesta. De cualquier forma, los pasos 1, 2 y 3 del proceso, serán ejecutados una sola vez cuando apuntamos a una nueva arquitectura por primera vez: en nuestro caso, la arquitectura móvil. En el capítulo 5 veremos el desarrollo de este proceso para obtener el ASM móvil.

Proceso de transformación. Basado en el enfoque MDA, llevado a cabo de forma iterativa y permitiendo un desarrollo incremental, este proceso implica pasos y actividades para transformar especificaciones a través de cada fase de MoWebA (ver figura 4.1):

- *CIM/PIM a ASM/PSM*, transformación M2M (*Model to Model*), hecho de manera (semi)automática, introduce algunos ajustes manuales, definiendo los metamodelos de la arquitectura o plataforma específica y las correspondientes reglas de transformación para esta fase;

- *ASM/PSM a ISM/Ajustes manuales*, transformación M2T (*Model to Text*), hecho de manera automática desde los modelos al código de la aplicación.

Algunas herramientas de transformación (de código abierto) utilizados por MoWebA son: Acceleo[24] y AndroMDA[25] [20].

4.3.2 Metamodelos y perfiles de MoWebA: Modelo de Entidades

González et al. [20] establecen los metamodelos y perfiles para cada uno de los diagramas del PIM de MoWebA mencionados en la sección 4.3.1. En particular, en la etapa 2 se especifica la estructura navegacional, los roles de usuario y el dominio. Para representar el dominio, MoWebA cuenta con el Diagrama de Entidades. Hablaremos de este diagrama en particular, puesto que será utilizado posteriormente en la definición de nuestra propuesta.

Representado por el Diagrama de Clases de UML, el Diagrama de Entidades define la estructura y las relaciones estáticas entre las clases identificadas en el dominio del problema. En la figura 4.4 presentamos el metamodelo y perfil de este diagrama, constituido por un conjunto de entidades (clases con el estereotipo *«entity»*). Cada clase posee propiedades y operaciones, con el atributo *visibility* para definir el tipo de visibilidad. Las entidades pueden conectarse entre sí a través de relaciones del tipo asociación, agregación y generalización.

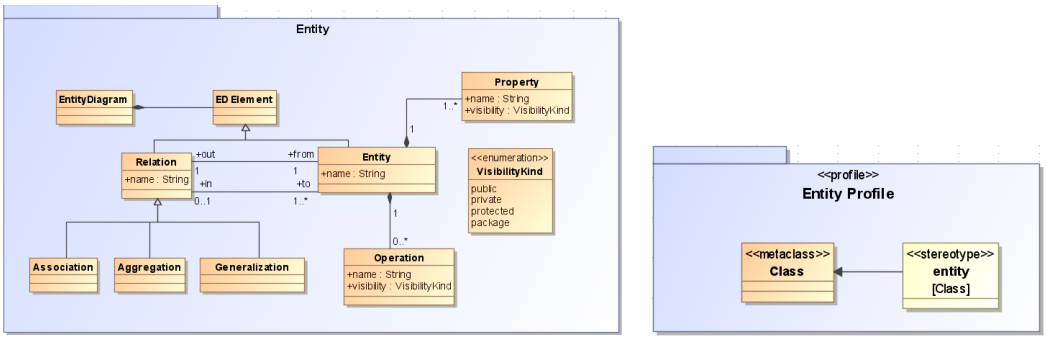


Figura 4.4 Metamodelo y Perfil UML del Modelo de Entidades

4.3.3 MoWebA y el desarrollo de aplicaciones móviles

En la sección 2.3, señalamos el problema de la fragmentación, la cual incrementa el esfuerzo de desarrollo y dificulta la portabilidad de las aplicaciones móviles ante tanta variedad de teléfonos móviles y plataformas móviles. Sanchiz et al. [53] resaltan y discuten

[24] Acceleo. http://www.eclipse.org/acceleo/
[25] AndroMDA. https://www.andromda.org/

ciertos aspectos de MoWebA que podrían tener impacto positivo en el problema de la portabilidad en el desarrollo de aplicaciones móviles. Ante un ambiente tan fragmentado, lograr avances hacia la portabilidad de estas aplicaciones para distintas plataformas sería de suma importancia, ahorrando esfuerzo de desarrollo y tiempo. A continuación nombramos estos aspectos:

(i) *La incorporación de ASM como una nueva etapa de modelado*.

MoWebA hace una clara distinción entre el espacio conceptual y los aspectos arquitecturales, definiendo modelos a nivel arquitectural: ASM. ASM enriquece los modelos con información específica para una arquitectura (RIAs, SOA, REST, arquitectura móvil, entre otros), lo que permite tener una flexibilidad de evolucionar y adaptarse a diferentes arquitecturas partiendo del mismo PIM (por ejemplo, una banca web RIA y su respectiva versión móvil). Un modelo ASM puede resultar en diferentes PSMs, dado que una misma arquitectura puede ser implementada en diferentes plataformas (por ejemplo, la arquitectura RIA puede ser implementada en diferentes plataformas como Backbase, Dojo, GWT, jQuery, entre otros) [20].

La posible contribución de este aspecto es la portabilidad del PIM con respecto a diferentes arquitecturas.

(ii) *Clara separación de la capa de presentación respecto a las capas de navegación y comportamiento*.

Ante tanta variedad de tamaño de pantallas de teléfonos móviles y aspectos de interfaz específicos por plataforma móvil (por ejemplo, estilo de letras, botones, distribución de elementos en la pantalla, entre otros), la capa de presentación presenta mayores dificultades. La clara separación de la capa de presentación de otras capas, permite tratar este problema e ir mejorando solo en esta capa, sin afectar a las otras.

La posible contribución de este aspecto es la portabilidad de la capa de presentación.

(iii) *Navegación más orientada a funciones que orientado a datos*.

En MoWebA, la navegación es definida considerando las unidades funcionales (Casos de Uso) como el nivel de granularidad, y los caminos de navegación son definidos considerando las relaciones entre los casos de uso, haciendo que la definición de la navegación se obtenga de la forma en que el usuario interactúa con el sistema [20]. Esto puede ayudar a obtener una estructura más comprensible, más clara, más fácilmente navegable y que refleja mejor cuáles son las necesidades del usuario.

La posible contribución de este aspecto es la portabilidad del modelado de navegación.

Los aspectos mencionados actúan en diferentes capas. Considerando las capas de MoWebA (ver figura 4.1), el punto (ii) actúa en la capa de presentación, y el punto (iii) en la capa de navegación. Por otra parte, el punto (i) habla de la incorporación del ASM como nueva etapa de modelado. El ASM constituye una fase de MoWebA que abarca todas las capas definidas. Como nuestra propuesta se centra en la capa de datos, creemos aprovechable el nivel ASM para la elaboración de nuestro enfoque de desarrollo de aplicaciones móviles nativas enfocadas en la capa de datos.

4.4 Síntesis del capítulo

En este capítulo se llevó a cabo un estudio cuyo objetivo principal fue obtener una visión global de MDD en el marco del desarrollo de las aplicaciones móviles nativas, teniendo en aspectos de persistencia en el desarrollo. Como resultado se obtuvo un total de 18 soluciones MDD para el desarrollo de aplicaciones móviles.

Del estudio se destacó la necesidad de especificaciones para describir la persistencia en el modelado de aplicaciones móviles y la poca adopción de MDA (en especial del estándar MOF). Se pretende cubrir conceptualmente la persistencia en el desarrollo de aplicaciones móviles, mediante elementos específicos que permitan tener en cuenta las distintas opciones y mecanismos de persistencia en el modelado, a fin de posibilitar el almacenamiento de datos incluso sin conexión de red, y considerando la conexión con fuentes remotas. La capa de datos cubre conceptualmente estos aspectos: persistencia y proveedores de datos.

MoWebA se presenta como una metodología MDD enfocada al desarrollo de aplicaciones web centrado en roles, bajo los estándares MDA. Ciertos aspectos de esta metodología podrían tener impacto positivo en el problema de la portabilidad en el desarrollo de aplicaciones móviles, en especial el ASM. El ASM constituye una fase de MoWebA que abarca todas las capas definidas. Como nuestra propuesta se centra en la capa de datos, creemos aprovechable el nivel ASM para la elaboración de nuestro enfoque de desarrollo de aplicaciones móviles nativas enfocadas en la capa de datos.

Capítulo 5

Una Extensión de MoWebA para el Desarrollo de Aplicaciones Móviles

En este capítulo se presenta nuestro enfoque MoWebA Mobile para el desarrollo de aplicaciones móviles nativas enfocadas en la capa de datos. Partimos de MoWebA, un enfoque MDD para el desarrollo de aplicaciones web, a partir del cual proponemos una extensión de su PIM para facilitar la introducción de elementos específicos de la arquitectura móvil en el ASM. A partir de las extensiones realizadas al metamodelo y perfil del diagrama de entidades, definimos un ASM para el desarrollo de aplicaciones móviles abarcando las funcionalidades y características mencionadas. Por último, desarrollamos reglas de transformación que permiten generar código de la aplicación móvil final a partir de los modelos definidos.

Las siguientes secciones se distribuyen como sigue: en la sección 5.1 se habla de los aspectos considerados por el enfoque y el proceso de desarrollo a partir del mismo. Luego, se define el ASM móvil (sección 5.2), se detallan las extensiones realizadas a MoWebA y se presenta el metamodelo y perfil móvil. A fin de mostrar un ejemplo de modelado con nuestra propuesta, en la sección 5.3 se presenta el modelado de una aplicación de compras *online*. Por último, en la sección 5.4 se presentan nuestras reglas de transformación y se detalla el código que generamos a partir de los modelos definidos.

5.1 MoWebA Mobile

Nuestra propuesta denominada MoWebA Mobile, parte de la extensión del metamodelo y perfiles de MoWebA. A partir de las extensiones al PIM de MoWebA, presentamos un ASM para móviles. El ASM propuesto se enfoca en el desarrollo de aplicaciones móviles

nativas enfocadas en la capa de datos. Como mencionamos en la sección 3.3, siguiendo la arquitectura MAAG, en este capa se consideran el almacenamiento local de datos, así como también los distintos proveedores de datos. *Las herramientas definidas para este enfoque (metamodelo, perfil y reglas de transformación) se encuentran disponible en la web del proyecto MDD+*[1].

En la sección 3.2.1 hemos presentado los distintos tipos de almacenamiento de datos en los teléfonos móviles inteligentes: base de datos, archivos y pares clave-valor. Hemos introducido elementos que permiten identificar qué datos van a almacenarse en el dispositivo (*persistentEntity*), y qué tipo de almacenamiento se utilizará (*persistentType*). La aplicación generada permitirá al usuario implementar estos tipos de almacenamiento, a partir de funcionalidades que facilitarán el manejo de los mismos.

En la sección 3.3 mencionamos distintas fuentes de datos que proporcionan datos a la aplicación móvil. Según las especificaciones del perfil de Riveiro y da Silva [50][51], las aplicaciones móviles pueden recibir datos de distintos proveedores de contenido: *XisServer* (representa un servidor remoto que interactúa con la aplicación), *XisClientMobileApp* (representa una aplicación que interactúa con la aplicación principal) y *XisInternalProviderApp* (representa un proveedor interno, que permite interactuar con las características nativas del sistema operativo como ser locación, contactos, calendario, media o *custom*). A partir de esto, identificamos que las aplicaciones móviles pueden recibir datos de tres tipos de fuentes en general: fuentes de datos externas (por ejemplo, servidores y base de datos externas), fuentes de datos internas (sensores como el giróscopo y el acelerómetro, y recursos de *hardware* específicos del dispositivo, como la cámara y el micrófono) y mediante la interoperabilidad con otras aplicaciones. Introducimos las siguientes interfaces: *WebServiceInterface*, *HardwareDeviceInterface* y *MobileAppDataInterface*, para la representación de las fuentes externas, internas y la interoperabilidad con otras aplicaciones, respectivamente. La comunicación con fuentes externas será mediante servicios web; definiremos una serie de sensores comunes con las plataformas que trabajaremos, con las que definiremos la comunicación con fuentes internas; y por último, definiremos los tipos de datos con los que trabajar al interoperar con otras aplicaciones.

En la figura 5.1 observamos el esquema general del proceso de desarrollo con MoWebA Mobile. Partimos del modelado con la herramienta MagicDraw[2] utilizando los perfiles de MoWebA y ASM móvil. Luego, a fin de generar código a partir de los modelos realizados, exportamos estos en formato XMI y los importamos en Acceleo. Utilizando reglas de transformación definidas (planillas de Acceleo), generamos el código fuente de la

[1] Herramientas MoWebA Mobile. http://www.dei.uc.edu.py/proyectos/mddplus/herramientas/mowebamobile/
[2] MagicDraw. https://www.nomagic.com/products/magicdraw.html

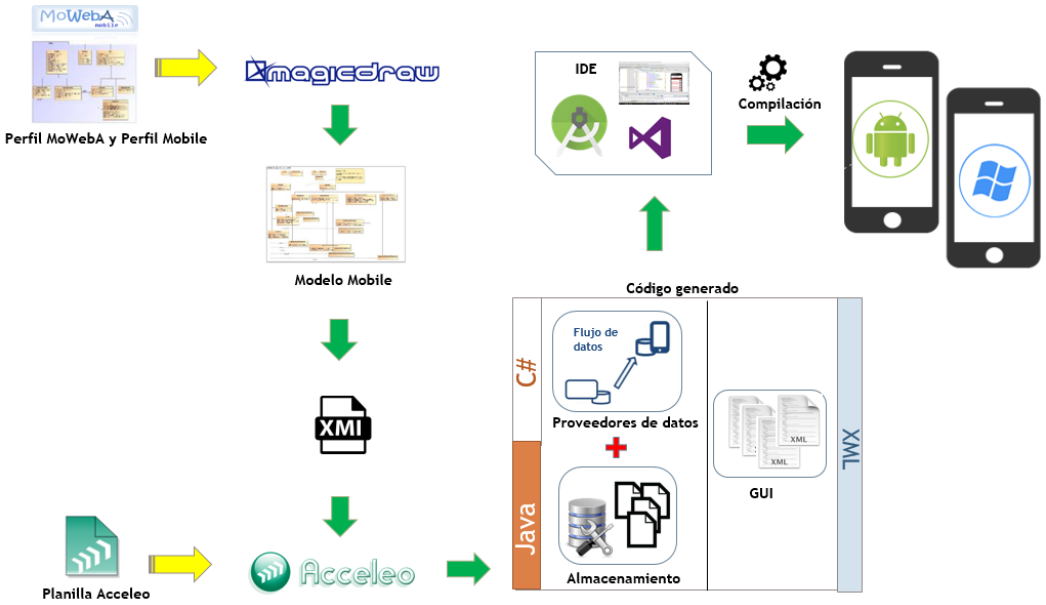


Figura 5.1 Esquema general del proceso de desarrollo con MoWebA Mobile

aplicación móvil. Para este PFC definimos reglas de transformación para generar aplicaciones demo, tanto para Android (código Java) como para Windows Phone (código C#), a fin de evidenciar las funcionalidades generadas por cada uno de los aspectos mencionados en nuestro enfoque. Por último, a fin de obtener la aplicación nativa para ambas plataformas, el código generado debe ser compilado en sus respectivos IDEs: Android Studio y Visual Studio. Las características de la aplicación generada serán detalladas en la sección 5.4.

En la siguiente sección hablamos en detalle sobre el proceso de definición del ASM móvil.

5.2 Definición del ASM móvil

Como mencionamos en la sección 4.3.3, pretendemos aprovechar el nivel ASM de MoWebA a fin de definir elementos específicos para la arquitectura móvil, que nos permita desarrollar aplicaciones móviles enfocadas en la capa de datos. En la sección 4.3.1 describimos el proceso de definición del ASM. En primer lugar, extendimos el PIM de entidades de MoWebA a fin de aprovechar elementos conceptuales del tipo estructural. Luego, definimos el metamodelo y perfil UML móvil a partir de las extensiones realizadas. Por último, como veremos en la sección 5.4, definimos las reglas de transformación para la obtención automática de los elementos definidos en el modelo, completando así el ciclo de definición del ASM.

5.2.1 Extensiones al PIM de MoWebA: Diagrama de Entidades

A fin de definir la estructura y las relaciones estáticas entre las clases identificadas en el dominio del problema [20], MoWebA cuenta con el Diagrama de Entidades. Extendimos este diagrama a fin de cubrir más elementos conceptuales del tipo estructural, que nos faciliten posteriormente en el modelado de aplicaciones móviles. La extensión propuesta se puede observar en la figura 5.2, donde presentamos el metamodelo y el perfil de entidades extendido. Con la intención de facilitar la comprensión de los cambios realizados, las clases en el metamodelo y en el perfil de entidades extendidos han sido coloreados, utilizando la siguiente convención de colores: en color salmón se encuentran las clases que no han sido alteradas de su forma original; en azul, las que han sido modificadas ya sea mediante la agregación, actualización o eliminación de alguna de sus propiedades; y por último, en verde, aquellas clases nuevas agregadas.

Para detallar los aspectos propuestos y extendidos, utilizamos el metamodelo extendido de entidades. Como vemos en la figura 5.2, se realizaron dos tipos de modificaciones: una extensión de las propiedades de una entidad (*EntityProperty*), y se agregaron los tipos de datos específicos que pueden tener estas propiedades (*dataType*). Las propiedades agregadas permiten especificar: el tipo de dato (*DataType*), cuyos valores pueden ser *bool*, *date*, *time*, *datetime*, *int*, *real*, *string* y *text*; un tamaño máximo de dato (*size*); restringir que el campo sea nulo (*notNull*); indicar un valor por defecto (*defaultValue*); y, establecer si la propiedad se constituye como identificador de la entidad (*id*).

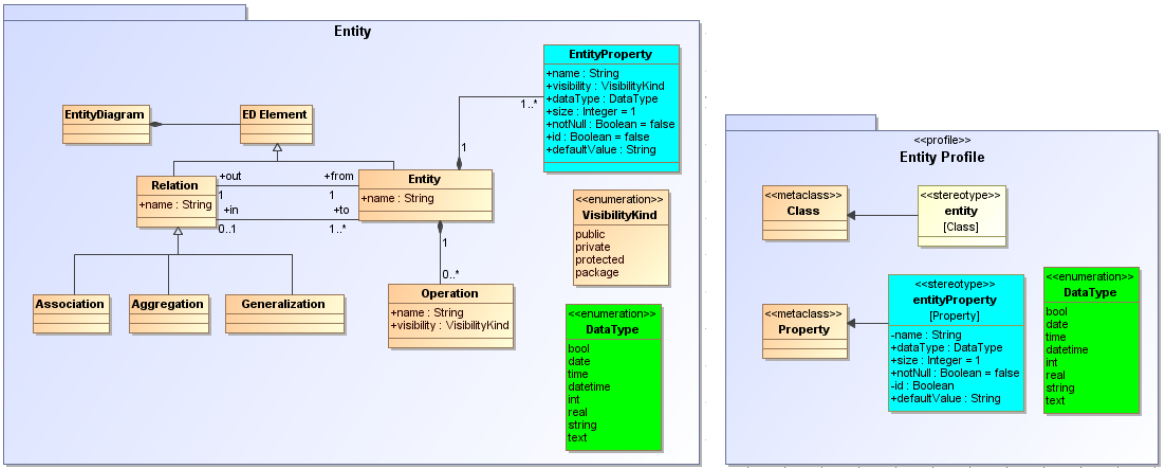


Figura 5.2 Metamodelo (*izquierda*) y Perfil (*derecha*) extendido de entidades

5.2.2 Metamodelo y perfil móvil

A partir de las extensiones realizadas al PIM de MoWebA, presentamos el modelo específico de la arquitectura móvil o ASM móvil. A partir de este modelo, podemos definir aplicaciones móviles con las funcionalidades establecidas: la persistencia local de datos y el diseño de los componentes para el acceso a datos. En la figura 5.3 observamos el metamodelo de nuestra propuesta para el desarrollo de aplicaciones móviles nativas enfocadas en la capa de datos a partir de MoWebA. De la figura podemos resaltar lo siguiente: se distinguen dos secciones en particular, la persistencia de datos (dentro del cuadro con líneas punteadas: *Persistent Data*) y los proveedores de datos (dentro del cuadro con líneas punteadas: *Data Provider*). En los proveedores de datos, encontramos tres subsecciones distinguibles: en rojo, identificamos otras aplicaciones como proveedores de datos (dentro del cuadro con líneas punteadas: *MobileAppData*); en rosado, identificamos las fuentes externas de datos (dentro del cuadro con líneas punteadas: *External Data*); y por último, en azul claro, las fuentes internas de datos (dentro del cuadro con líneas punteadas: *Internal Data*). En la figura 5.4 observamos el perfil correspondiente.

A continuación pasamos a desarrollar cada uno de los aspectos mencionados: persistencia de datos y proveedores de datos.

Persistencia de datos

Utilizamos el diagrama de persistencia para definir nuestras entidades persistentes (*persistentEntity*), con los cuales especificamos el tipo de almacenamiento que deseamos utilizar en nuestra aplicación y así poder manejar nuestros datos. De igual forma, agregamos propiedades a estas entidades persistentes (*persistentEntityProperty*) para enriquecer conceptualmente nuestro modelo.

La entidad persistente puede determinarse con qué tipo de almacenamiento será almacenado gracias al valor etiquetado asociado (*persistentType*), el cual nos brinda las siguientes opciones: *Database*, el cual nos permite generar una base de datos y utilizar el nombre de la entidad persistente como una tabla; *File*, nos permite guardar la entidad persistente en archivos, mediante funciones que facilitan el manejo de archivos (por ejemplo, leer, escribir y otros); y, por último, *KeyValue*, el cual nos permite almacenar la entidad persistente en pares clave-valor, mediante funciones que proporcionamos para el manejo de estos datos.

Cada atributo de una entidad persistente posee propiedades con estereotipo *«PersistentEntityProperty»*. Cada propiedad a su vez puede ser enriquecida con subpropiedades, definidas como valores etiquetados. Tenemos las propiedades definidas en

el PIM de entidades (ver sección 5.2.1). A parte de los mencionados, definimos la propiedad *selectable*, utilizado para la persistencia con bases de datos (*persistentType = Database*). Esta propiedad nos permite indicar si un atributo de la entidad persistente será usado como valor clave de selección de datos, para realizar las operaciones de borrado y actualización en las base de datos.

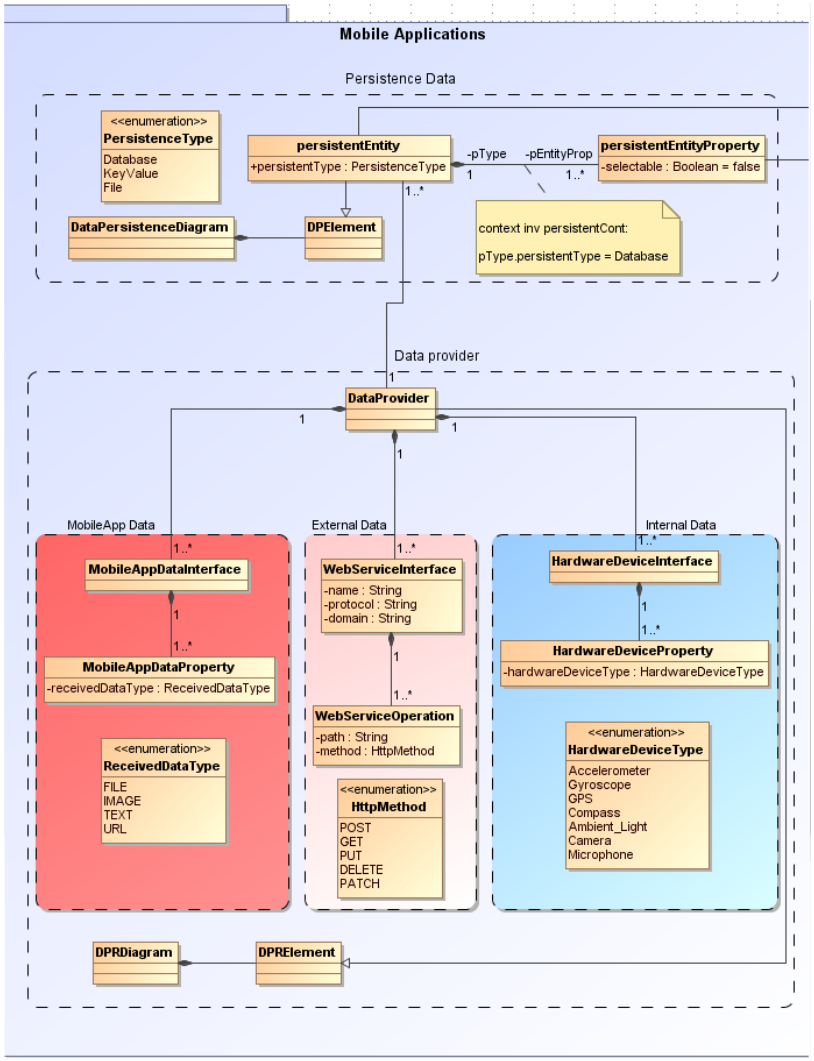


Figura 5.3 Metamodelo de MoWebA Mobile

Por último, incluimos el estereotipo *«DataPersistence»* para identificar el paquete donde será incluido el modelo de datos persistentes. Al momento de trabajar con las reglas de transformación y poder generar código a partir de este modelo, esta propiedad nos permite identificar que estamos trabajando con el modelo de datos persistentes. En la sección 5.4.2 detallaremos el código que se genera a partir de este modelo.

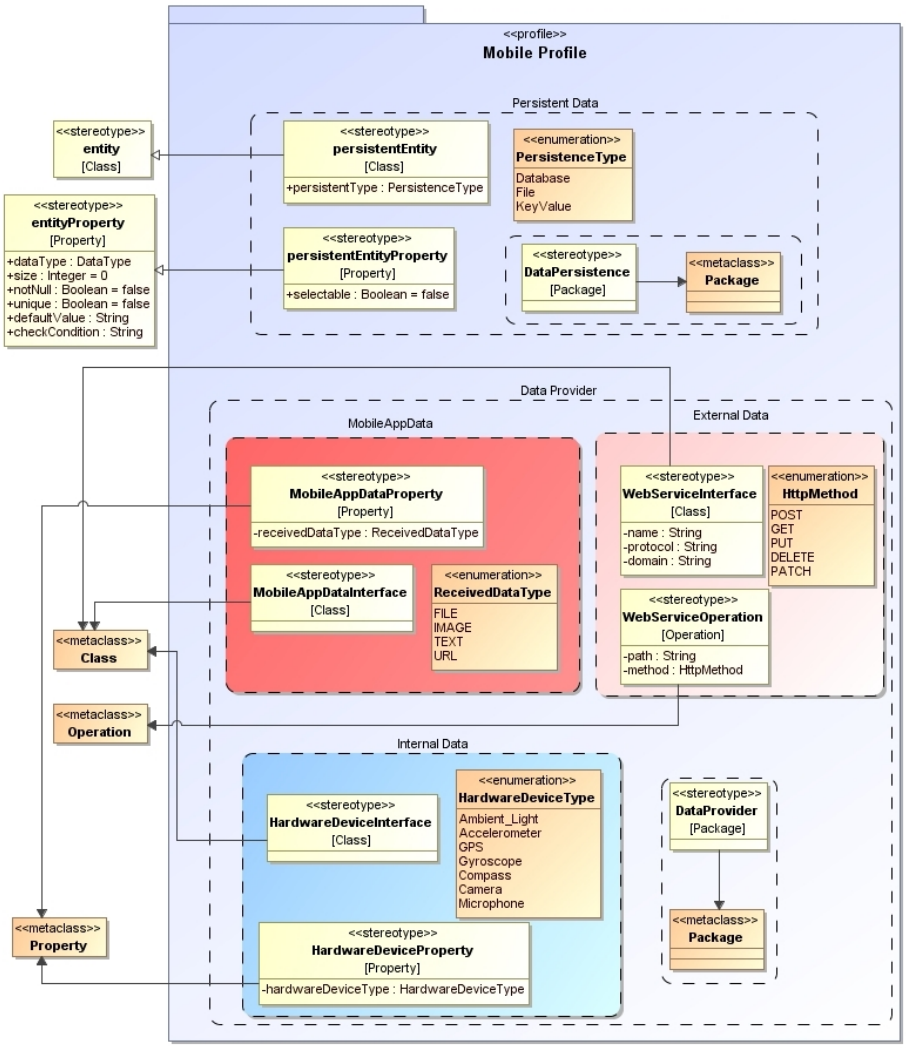


Figura 5.4 Perfil de MoWebA Mobile

Proveedores de datos

Anteriormente mencionamos que una aplicación puede recibir datos de tres tipos de fuentes: interna, externa y mediante otras aplicaciones instaladas en el mismo dispositivo. Utilizamos el diagrama de proveedores de datos, compuesto por elementos específicos para poder diseñar los proveedores de datos para la aplicación móvil.

Los proveedores externos proporcionan datos mediante la comunicación de la aplicación móvil con servidores o base de datos externas. Esta comunicación la realizamos mediante servicios web siguiendo la arquitectura REST. La clase *WebServiceInterface*, mediante valores etiquetados, permite definir la URL base de conexión: protocolo y dominio ([*protocol*]:[*domain*]). Con *WebServiceOperation* definimos las funciones o servicios en

nuestra clase *WebServiceInterface*. El nombre de la función corresponde con el nombre del servicio, al cual, mediante valores etiquetados, le agregamos el método http que utiliza para la conexión (*method*) con posibles valores: *POST*, *GET*, *PUT*, *DELETE* y *PATCH*; y la ruta de acceso al servicio (*path*).

Los teléfonos móviles inteligentes cuentan con sensores y con recursos de *hardware* específicos (por ejemplo, cámara, micrófono, entre otros) que proveen flujos de datos; consideramos estos como proveedores internos de datos. La clase *HardwareDeviceInterface* nos permite definir qué sensores y/o recursos de *hardware* propios del dispositivo utilizar mediante *HardwareDeviceProperty* y su valor etiquetado *hardwareDeviceType*. Como posibles opciones tenemos: acelerómetro (*Accelerometer*), giróscopo (*Gyroscope*), geolocalización (*GPS*), brújula (*Compass*), sensor de luz (*AmbientLight*), cámara (*Camera*) y micrófono (*Microphone*).

De igual forma, la aplicación móvil puede interoperar con otras aplicaciones instaladas en el mismo dispositivo móvil, enviando o recibiendo datos. Mediante la clase *MobileAppDataInterface* podemos representar aquellos datos que la aplicación móvil recibe de otras aplicaciones. Con el valor etiquetado *ReceivedDataType* de la propiedad (*MobileAppDataProperty*), seleccionamos qué tipo de dato será capaz de recibir y manejar la aplicación. Como opciones encontramos: *File*, para intercambio de archivos en general; *Image*, para recibir imágenes; *Text*, para recibir texto; y *Url*, para recibir enlaces (*links*) en general.

Por último, incluimos el estereotipo *«DataProvider»* para identificar el paquete donde será incluido el modelo de proveedores de datos. Al momento de trabajar con las reglas de transformación y poder generar código a partir de este modelo, esta propiedad nos permite identificar que estamos trabajando con el modelo de proveedores de datos. En la sección 5.4.2 detallaremos el código que se genera a partir de este modelo.

5.3 Modelo de una aplicación móvil

Presentamos un ejemplo de modelado de una aplicación móvil llamado *e-market*, partiendo del ASM móvil, enfocados en los aspectos mencionados. Siguiendo el proceso de desarrollo descrito en la figura 5.1, utilizamos la herramienta MagicDraw para realizar el modelado de la aplicación, utilizando el perfil ASM móvil definido.

La aplicación consiste en una tienda virtual que hace entregas a domicilio. Uno tiene disponible un catálogo de productos y toda la información necesaria de cada producto. Si se desea adquirir un producto en particular, se selecciona dicho producto y se agrega al carrito de compras. Una vez finalizado la selección de los productos, se finaliza la compra, con la

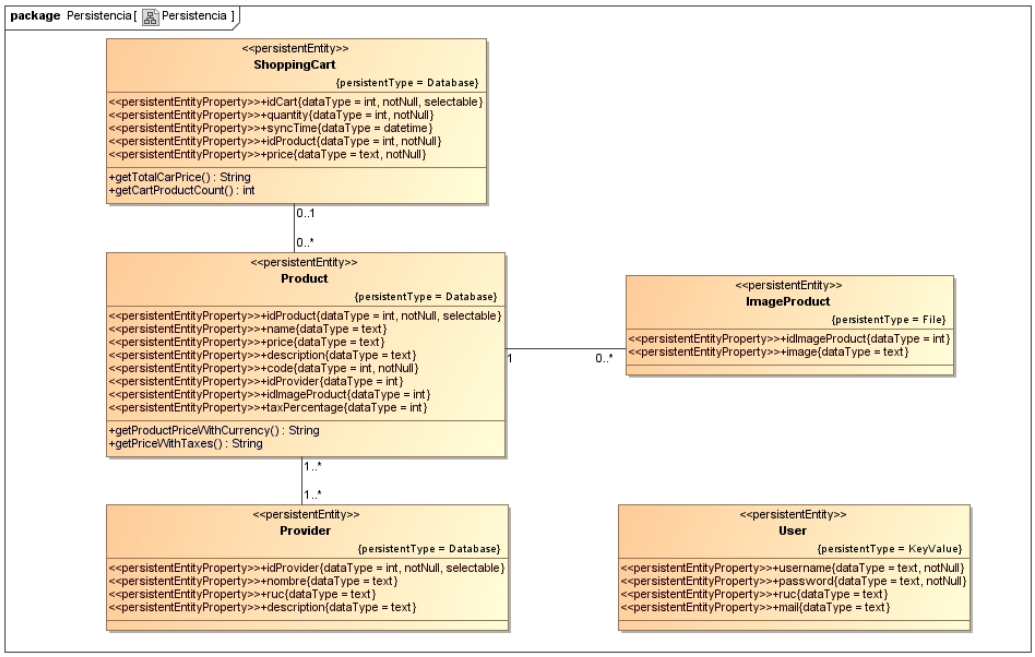


Figura 5.5 Modelo de la persistencia de datos de la aplicación *e-market*

posibilidad de indicar si se desea que la entrega se haga a domicilio o no. El usuario necesita estar registrado para ingresar a la aplicación. La aplicación debe estar disponible en todo momento, aún sin contar con una conexión permanente a Internet (modo "sin conexión").

En la figura 5.5 vemos el diseño de la estructura de datos de nuestra aplicación. Uno de los requisitos fue que la aplicación esté disponible aún sin conexión de red, por lo que utilizamos las entidades persistentes (*persistentEntity*) para indicar dónde almacenaremos los datos. Tenemos cinco entidades persistentes, cada una con propiedades estereotipadas con *«persistentEntityProperty»*: un producto (*Product*) tiene uno o más proveedores (*Provider*), y a la vez puede tener cero o más imágenes asociadas. Tenemos un carrito de compras (*ShoppingCart*) donde iremos almacenando los productos que se elijan, el cual puede estar compuesto de cero o más productos. Por último, tenemos a un usuario (*User*) donde almacenamos los datos de registro a la aplicación. Como observamos, cada entidad persistente posee un tipo de persistencia específico (*persistentType*): tanto los productos, el carrito de compras y los proveedores serán almacenados en una base de datos local (*persistentType = Database*), por lo que éstos tienen definido un atributo con el valor etiquetado *selectable*. Las imágenes de cada producto serán almacenados en forma de archivos en una carpeta local del dispositivo (*persistentType = File*). La sesión del usuario, y sus datos, los almacenaremos en pares clave-valor (*persistentType = KeyValue*).

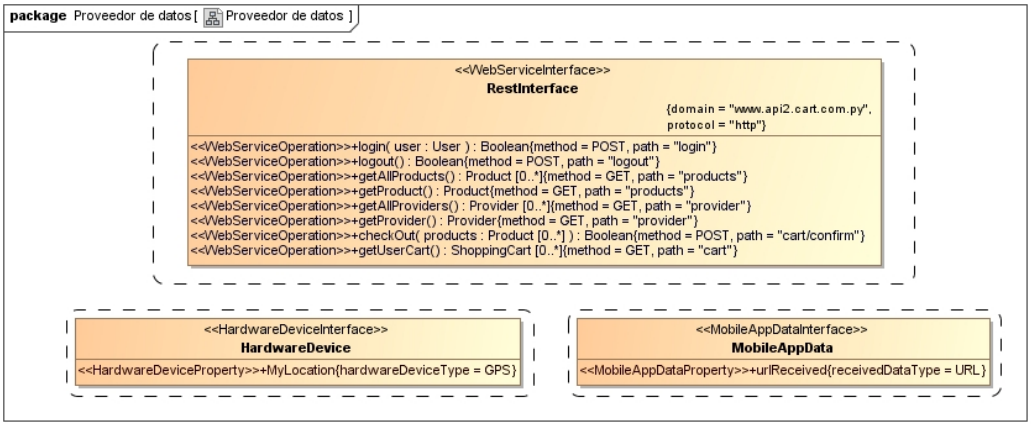


Figura 5.6 Modelo de los proveedores de datos de la aplicación *e-market*

Para representar de dónde obtendremos estos datos, modelamos los proveedores de datos, como vemos en la figura 5.6. Tenemos la clase *RestInterface* con estereotipo *«WebServiceInterface»* donde definimos las interfaces de los servicios que estaríamos utilizando. Con valores etiquetados definimos la URL base: *domain* y *protocol*. Para representar los servicios, se usan las operaciones con el estereotipo *«WebServiceOperation»*, donde usamos el nombre del mismo como nombre del servicio, y con valores etiquetados definimos a qué método http corresponde (*method*) y la ruta al servicio (*path*). Por otra parte, creamos la clase *HardwareDevice* con estereotipo *«HardwareDeviceInterface»*. Aquí definimos el atributo *MyLocation*, con estereotipo *«HardwareDeviceProperty»*, e indicamos que estaríamos recibiendo los datos de localización del dispositivo mediante GPS (*hardwareDeviceType = GPS*) para obtener nuestra dirección en caso de la entrega a domicilio. Por último, la clase *MobileAppData* con estereotipo *«MobileDataInterface»*, nos permite definir que la aplicación podrá ser capaz de recibir datos compartidos del tipo URL (*receivedType = URL*). Mediante esto, cualquier otra aplicación, podrá interactuar con esta aplicación y realizar determinadas acciones en respuesta a eso. Por ejemplo, como usuario uno puede recibir una promoción de un determinado producto en el correo; al dar clic al enlace, se abriría la aplicación y nos mostraría el producto listo para agregarlo a nuestro carrito de compras. Esta es una funcionalidad extra que nos pareció interesante mostrar en el modelado.

Un aspecto importante a tener en cuenta en el modelado es el manejo de la estructura de carpetas del proyecto de modelado. En la figura 5.7 podemos observar que la carpeta raíz del proyecto es *"CarritoDeCompras"*. Dentro encontramos dos carpetas, una para cada modelo. Podemos ver que la carpeta *"Persistencia"* tiene el estereotipo *«DataPersistence»*, y que la carpeta *"Proveedor de datos"* posee el estereotipo *«DataProvider»*. A fin de generar código

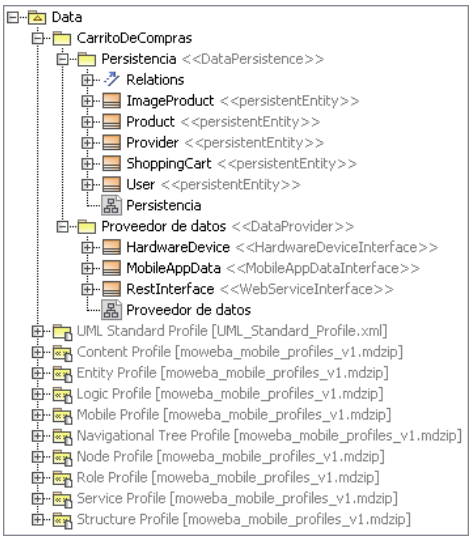


Figura 5.7 Estructura de carpetas del proyecto en MagicDraw del modelado de la aplicación *e-market*

a partir de estos modelos, mediante las reglas de transformación, estos estereotipos ayudan a distinguir entre los modelos proporcionados.

5.4 Reglas de transformación

A partir del modelado con el ASM móvil propuesto, generamos automáticamente una aplicación móvil demo, con el objetivo de evidenciar las funcionalidades generadas a partir de cada uno de los aspectos mencionados. En la figura 5.8 observamos la estructura general de la aplicación generada. Esta aplicación es completa, puede ser instalada en el teléfono móvil, funcional para las plataformas Android y Windows Phone. Como mencionamos en la sección 2.1.1, Android lidera la lista de sistemas operativos móviles más usados y Windows Phone queda en tercer lugar. Por una parte, decidimos desarrollar para Android por ser el sistema operativo más utilizado; por otra parte, elegimos Windows Phone porque a partir de los resultados obtenidos del estudio de propuestas MDD (ver sección 4.2.3) notamos que la mayoría de las propuestas generaban para Android y iOS, dejando Windows Phone muy desaprovechado para las pruebas.

Respecto a la persistencia de datos, basados en la tabla 3.1 utilizamos los mecanismos de persistencia específicos para estas plataformas: la base de datos generada es SQLite para ambas plataformas (en la sección 3.2.3 vimos que es la base de datos móvil más utilizada actualmente); los archivos se guardan en el dispositivo, se generan funciones específicas para cada plataforma que nos permiten realizar operaciones de lectura y escritura; y por último,

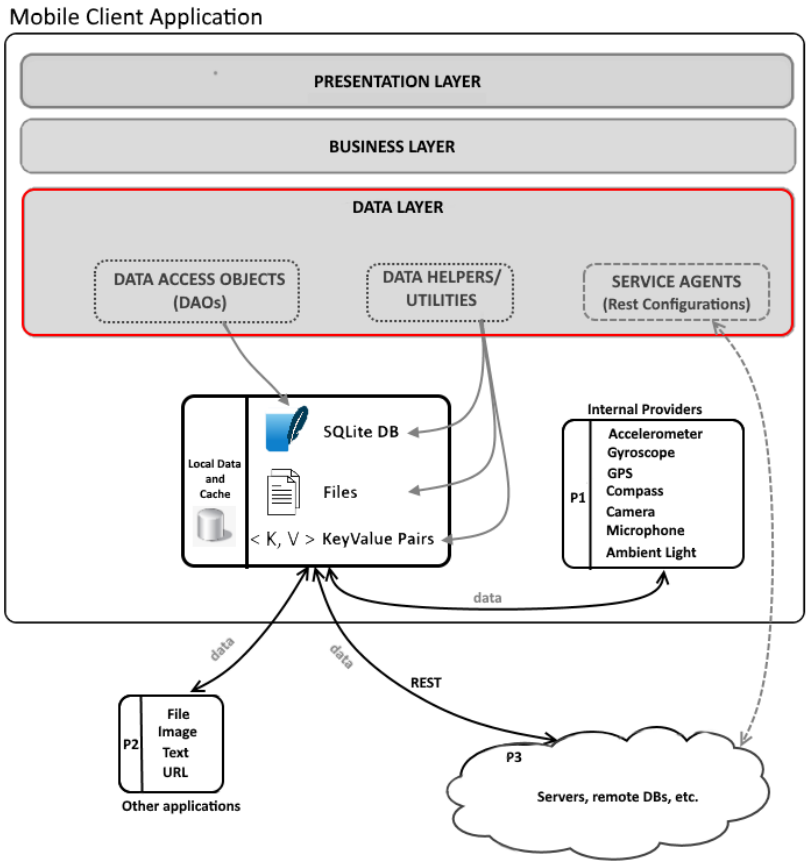


Figura 5.8 Estructura de la aplicación generada con MoWebA Mobile

los pares clave-valor permiten el manejo de los *SharedPreferences* y del *LocalStorage*, para Android y Windows Phone respectivamente.

Generamos pantallas que nos permiten hacer pruebas de los distintos mecanismos de persistencia definidos. Son formularios definidos a partir de cada entidad persistente que permiten cargar datos y realizar distintas operaciones sobre estos (por ejemplo, agregar, actualizar y borrar).

Respecto a los proveedores de datos, definimos: la comunicación con proveedores externos (servidores o base de datos externas) la realizamos mediante servicios web siguiendo la arquitectura REST; identificamos una serie de opciones de sensores y recursos de *hardware*, comunes en los teléfonos móviles inteligentes: acelerómetro (*Accelerometer*), giróscopo (*Gyroscope*), *GPS*, brújula (*Compass*), sensor de luz (*AmbientLight*), cámara (*Camera*) y micrófono (*Microphone*); y por último, identificamos distintos tipos de datos comúnmente utilizados en el intercambio de datos entre aplicaciones (*ReceivedDataType*) como ser *File* (para intercambio de archivos en general), *Image* (para recibir imágenes), *Text* (para recibir texto), y *Url* (para recibir enlaces o *links*).

Generamos una pantalla que permite verificar los valores arrojados por cada uno de los sensores y probar los recursos de *hardware* del dispositivo (como el GPS y la cámara) indicados.

Además de esta aplicación demo, proporcionamos clases auxiliares (o *helpers*), brindando al usuario funcionalidades que le permitirán manejar los distintos aspectos mencionados.

A partir del modelo de la aplicación *e-market* presentado en la sección 5.3, y considerando los aspectos de la aplicación a generar mencionados previamente, procedemos a generar código para las plataformas Android y Windows Phone. Siguiendo el proceso de desarrollo descrito en la figura 5.1, exportamos los modelos de MagicDraw en formato XMI, y luego los importamos a Acceleo. En la figura 5.9 observamos el *main template* de las reglas de transformación en Acceleo, el cual describe el siguiente proceso de generación de código:

- Se empieza por generar los archivos de configuración de la aplicación. En estos se otorgan los permisos necesarios para utilizar los distintos elementos del teléfono, y constituyen la estructura base de la aplicación, sobre la cual se van definiendo las demás clases. Además, aquí se tiene en cuenta la generación de uno de los proveedores de datos: interoperabilidad con otras aplicaciones. Para permitir esto, se necesitan permisos especiales para la aplicación, por eso aquí ya se tiene en cuenta este aspecto. Se busca la clase con el estereotipo *«MobileAppDataInterface»* dentro del paquete *«DataProvider»*.

- Luego, se recorren todos los paquetes en búsqueda de aquellos con los estereotipos *«DataPersistence»* y *«DataProvider»*.

  - En el primer caso, al encontrar *«DataPersistence»*, generamos los modelos de objetos. Luego, preguntamos por los *'persistentType'* definidos: si es *'Database'*, generamos código para crear y manejar base de datos; y tanto para *'File'* y *'KeyValue'* generamos las clases *helpers*, con todas las funcionalidades necesarias para el manejo de archivos y de los pares clave-valor, respectivamente. Veremos más detalles en la sección 5.4.2.

  - En el segundo caso, al encontrar *«DataProvider»*, buscamos entre las clases dentro del paquete aquellas con el estereotipo *«WebServiceInterface»* y *«HardwareDeviceInterface»*, para generar código para las interfaces de los servicios REST y para el manejo de los sensores definidos, respectivamente. Veremos más detalles en la sección 5.4.2.

En las siguientes secciones entraremos en detalle a analizar el código generado: la estructura de carpetas para cada plataforma y las funcionalidades generadas a partir de cada

modelo. En el Anexo B encontramos más detalles de la aplicación generada, pantallas y código generado.

```
[template public generateElement(model: Model)]
[comment @main/]
[let aPackages: Sequence(Package) = model.eAllContents(Package) ]

[generateGeneralAndroidClasses(model)/]
[generateGeneralWindowsClasses(model)/]

[for (aPackage : Package | aPackages)]
[let aClasses: Set(Class) = aPackage.ownedElement->filter(Class) ]
[let p : Package = aPackage.ancestors(Package)->first()]
[comment]Recorremos los paquetes existentes en el modelo. Solo nos interesa dos paquetes:
 DataPersistence y DataProvider[/comment]

    [comment]Si existe el paquete DataPersistence[/comment]
    [if (aPackage.hasStereotype('DataPersistence'))]

        [comment]Genera los beans o modelos de la aplicacion[/comment]
        [beansGenAndroid(aPackage, p.name.toLower())/]
        [beansGenWindows(aPackage)/]

        [comment]Solo creamos este archivo Si existe por lo menos una entidad tipo Database[/comment]
        [if (aPackage.isPackageHasThisPropertyStereotype('persistentEntity', 'persistentType', 'Databas
            [generateDBForAndroid(aPackage, p.name.toLower())/]
            [generateDBForWindows(aPackage, p.name.toUpperFirst())/]
        [/if]

        [comment]Solo creamos este archivo Si existe por lo menos una entidad tipo File[/comment]
        [if (aPackage.isPackageHasThisPropertyStereotype('persistentEntity', 'persistentType', 'File'))
            [generateFilesForAndroid(aPackage, p.name.toLower())/]
            [generateFilesForWindows(aPackage, p.name.toUpperFirst())/]
        [/if]

        [comment]Solo creamos este archivo Si existe por lo menos una entidad tipo KeyValue[/comment]
        [if (aPackage.isPackageHasThisPropertyStereotype('persistentEntity', 'persistentType', 'KeyValu
            [generateKVForAndroid(aPackage, p.name.toLower())/]
            [generateKVForWindows(aPackage, p.name.toUpperFirst())/]
        [/if]
    [/if]

    [comment]Si existe el paquete DataProvider[/comment]
    [if (aPackage.hasStereotype('DataProvider'))]
        [for (aClass : Class | aClasses)]
            [comment]Si la clase tiene el paquete WebServiceInterface[/comment]
            [if (aClass.hasStereotype('WebServiceInterface'))]
                [generateRestAndroid(aClass, p.name.toLower())/]
                [generateRestWindows(aClass, p.name.toUpperFirst())/]
            [/if]

            [comment]Si la clase tiene el paquete HardwareDeviceInterface[/comment]
            [if (aClass.hasStereotype('HardwareDeviceInterface'))]
                [generateSensorsAndroid(aClass, p.name.toLower())/]
                [generateSensorsWindows(aClass, p.name.toUpperFirst())/]
            [/if]
        [/for]
    [/if]
[/let]
[/let]
[/for]
[/let]
[/template]
```

Figura 5.9 *Template* principal de las reglas de transformación en Acceleo

5.4.1 Estructura de carpetas del código generado

Se generan dos estructuras de carpetas, una para Android (*android/*) y otra para Windows Phone (*windows/*). Cada plataforma presenta una estructura de carpetas diferente (ver figura 5.10), a continuación brindaremos detalles de cada una.

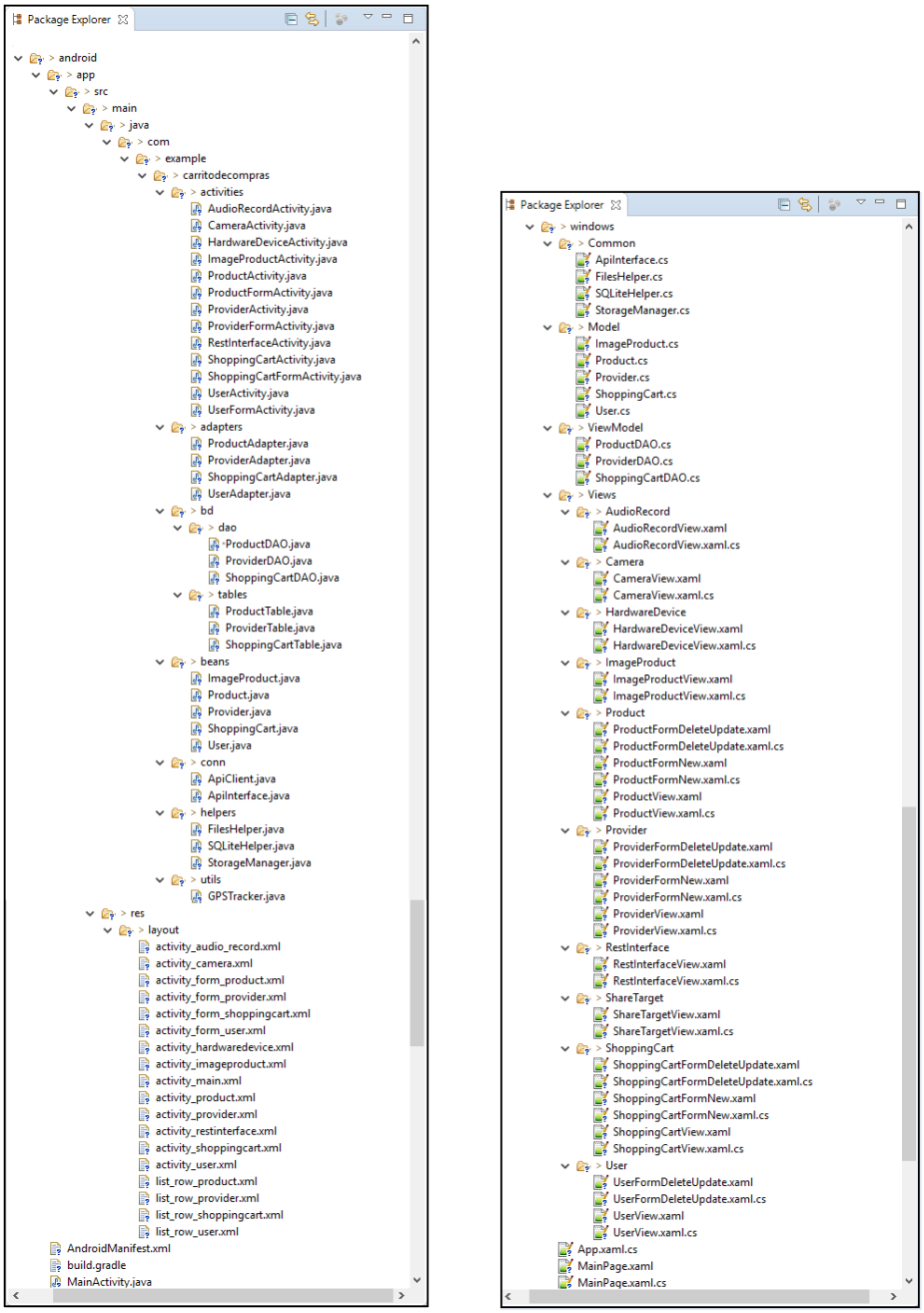


Figura 5.10 Estructura de directorios generada para las plataformas móviles. Aplicación *e-market*. *A la izquierda Android (java), a la derecha Windows Phone (C#)*

Estructura de carpetas para Android

La estructura de carpetas generada para Android, sigue el patrón de diseño de los proyectos en Android Studio. En la página del desarrollador encontramos la Guía del Usuario[3], donde se brindan más detalles de la estructura utilizada.

[3] *Android User Guide*. https://developer.android.com/studio/projects/index.html

Dentro de la estructura de carpetas, destacamos dos sub-carpetas: *java* (*app/src/main/java*) donde se encuentra el código fuente en *java*, y *res* (*app/src/main/res*) encontramos los recursos de la Interfaz de Usuario (UI) de la aplicación.

En *java/* encontramos la estructura de carpetas a partir del nombre del paquete de la aplicación, en nuestro caso el nombre es fijo: *com.example.<nombre del proyecto de modelado>*, donde el *<nombre del proyecto de modelado>* se refiere al nombre de la carpeta raíz del proyecto de modelado. Como vimos en la figura 5.7, para la aplicación *e-market*, la carpeta raíz del proyecto es *"CarritoDeCompras"*. En *app/src/main/java/com/example/carritodecompras/* encontramos el código *java*, dispuesto en las siguientes subcarpetas:

- *activities/* : contiene todos los *activities*[4] de la aplicación. Se definen las pantallas y la lógica de la misma.
- *adapters/* : contiene los adaptadores o *adapters*[5] de la aplicación, donde se especifican los datos que contendrán los *Views*. En nuestro caso, los adaptadores nos ayudan a definir los campos que contendrá cada ítem de la lista de datos, tanto de la base de datos como de los pares clave-valor.
- *bd/* : agrupa todas las funcionalidades referentes a las base de datos (la definición de las tablas y los *Data Access Objects*[6] o DAOs).
- *beans/* : contiene los modelos del dominio de la aplicación.
- *conn/* : maneja la conexión a red de la aplicación.
- *helpers/* : contiene las clases auxiliares con funcionalidades específicas para cada aspecto de persistencia mencionado.
- *utils/* : contiene aquellas clases con funcionalidades útiles y de carácter general, para toda la aplicación.

En *res/* encontramos la carpeta *layout/*, con los archivos de diseño de la UI de la aplicación (**.xml*). Por cada clase definida en *activities/* y en *adapter/*, se genera un archivo *xml*.

Respecto a los archivos de configuración de la aplicación, en *android/* generamos los siguientes archivos: *AndroidManifest.xml*, *build.gradle* y el *MainActivity.java*.

[4] Los *activities* sirven como punto de entrada de interacción del usuario con la aplicación, y define la navegación del usuario en la aplicación. *Fuente: https://developer.android.com/guide/components/activities/index.html*

[5] *Adapters*. https://developer.android.com/reference/android/widget/Adapter.html

[6] *Core J2EE Patterns - Data Access Object*. http://www.oracle.com/technetwork/java/dataaccessobject-138824.html

Estructura de carpetas para Windows Phone

Siguiendo las *Prácticas & Patrones de Diseño de Aplicaciones Móviles de Microsoft*[7], la estructura de carpetas para Windows Phone sigue el patrón de diseño *Model-View-ViewModel*[8] o MVVM.

En *windows/* encontramos la siguiente estructura de carpetas:

- *Common/* : una carpeta para los componentes que se comparten entre aplicaciones.
- *Model/* : contiene los modelos del dominio de la aplicación.
- *ViewModel/* : contiene todas las clases responsables de la lógica de las vistas.
- *Views/* : contiene las vistas de la aplicación, responsables de definir la estructura y el diseño de las pantallas.

El nombre de la aplicación generada es el <*nombre del proyecto de modelado*>, y como vimos se refiere al nombre de la carpeta raíz del proyecto de modelado en este ejemplo: *"CarritoDeCompras"* y constituye el *namespace* raíz del proyecto.

5.4.2 Arquitectura del código generado

Se consideran tres aspectos principales en la generación de código: archivos de configuración, la persistencia y los proveedores de datos. Los archivos de configuración contienen las declaraciones de las clases definidas en la aplicación, así como los permisos requeridos para su utilización. Además, constituyen la estructura base de la aplicación, sobre la cual se van definiendo las demás clases. Respecto a la persistencia, se generan las clases de utilidades y *helpers* a disposición del usuario, y las funciones que permiten el manejo de los mecanismos de persistencia definidos. Por último, se generan las clases necesarias para la configuración, manejo y acceso a los proveedores de datos definidos.

Archivos de configuración de la aplicación

Para toda aplicación creamos clases principales, que se encargan de la configuración de la aplicación, tanto de los permisos y de las pantallas declaradas, así como de la definición de la pantalla principal de la aplicación.

La configuración de la aplicación y la generación de código son diferente para cada plataforma. Para Android se generan los siguientes archivos:

[7] *Developing a Windows Phone Application using the MVVM Pattern.* https://msdn.microsoft.com/en-us/library/hh848247.aspx

[8] *The MVVM Pattern.* https://msdn.microsoft.com/en-us/library/hh848246.aspx#MVVMBenefits

- El *MainActivity.java* y su layout *xml*, constituyen la pantalla principal de la aplicación.
- El archivo de configuración *AndroidManifest.xml*, donde se definen las pantallas que contendrá la aplicación, y los permisos que necesitará para utilizar las distintas funcionalidades definidas.
- El *build.gradle*, encargado de manejar e importar las librerías necesarias para la aplicación.

Por otra parte, para Windows Phone generamos:

- El *App.xaml.cs*, constituye el archivo de entrada de la aplicación.
- El *MainPage.xaml.cs* y su interfaz (*MainPage.xaml*), constituyen la pantalla principal de la aplicación.

En la figura 5.11 observamos la pantalla principal de la aplicación generada en ambas plataformas.

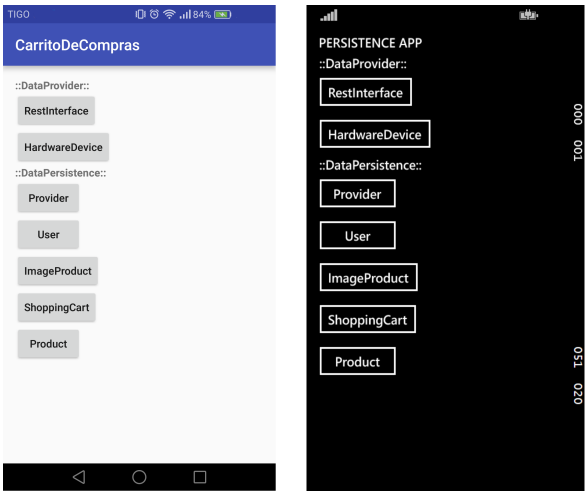


Figura 5.11 Menú principal. Aplicación *e-market*. *A la izquierda Android, a la derecha Windows Phone*

Persistencia de datos

A partir del modelo de persistencia de datos (figura 5.5), utilizamos las siguientes reglas de transformación para generar código: modelos de objetos, manejo de base de datos, manejo de los archivos y de los pares clave-valor.

Código generado: Modelo del dominio

Según el patrón MVVM, el modelo es una implementación del modelo del dominio de la aplicación, que incluye el modelo de datos junto con la lógica de negocio. Para Android, generamos *JavaBeans*[9] (o simplemente *Beans*), los cuales constituyen los modelos de datos. Siguiendo el patrón MVVM para Windows Phone, generamos modelos (*Models*).

El proceso de generación para ambas plataformas fue la misma: por cada clase *«persistentEntity»* declarada en el modelo de persistencia, se genera un modelo de datos; y los atributos que manejan estos modelos, corresponden a los *«persistentEntityProperty»* declarados en cada entidad persistente. Cada atributo posee su *getter* y *setter*, así como los constructores de la clase. De la misma forma, se generan las funciones declaradas en las entidades persistentes (*operations*).

Código generado: Base de datos

Generamos código SQLite tanto para Android como para Windows Phone, que nos permite crear la base de datos, junto con sus tablas y las funciones CRUD para manejar los datos.

Por cada clase *«persistentEntity»*, con el valor etiquetado *persistentType* igual a *"Database"*, generamos los siguientes elementos:

- **Formulario de datos**. Generamos las pantallas para listar, agregar, editar y eliminar datos.

  En Android, estas pantallas se definen en *activities/*. Por cada entidad persistente generamos dos clases *java* (cada una corresponde a una pantalla de la aplicación) y sus correspondientes archivos UI (*.xml*):

  - *<nombre de la clase>Activity*: listan los datos guardados correspondientes a la entidad persistente, con la posibilidad de eliminarlos y editarlos. Cada dato de esta lista corresponde a una fila de la tabla de la base de datos. Para generar la estructura de cada "fila" de la tabla a listar, usamos adaptadores, donde definimos cada campo que lo compone. Por cada entidad persistente se define un adaptador (*adapters/*).
  - *<nombre de la clase>FormActivity*: formularios para la carga y edición de los datos almacenados.

[9] *JavaBeans*. http://docs.oracle.com/javase/tutorial/javabeans/

En Windows Phone estas pantallas son definidas en *Views/*, donde por cada entidad persistente se generan tres archivos *.xaml* (cada una corresponde a una pantalla de la aplicación) y sus respectivos archivos UI (*.xaml.cs*):

  - *<nombre de la clase>View*: listan los datos guardados correspondientes a la entidad persistente, con la posibilidad de eliminarlos y editarlos. Cada dato de esta lista corresponde a una fila de la tabla de la base de datos.
  - *<nombre de la clase>FormNew* y *<nombre de la clase>FormDeleteUpdate*: formularios para la carga, edición y eliminación de los datos almacenados.

- **Funciones CRUD para el manejo de los datos**. El acceso a datos varía según la fuente de datos, usamos los DAOs para encapsular y abstraer todos los accesos a la base de datos. En la estructura de carpetas generadas para Android, encontramos los DAOs en *bd/dao*. En Windows Phone, son los *ViewModels* (*ViewModel/*). Las funciones generadas en cada DAO son:

  - *add<nombre de la clase>Object)*: permite insertar un objeto a la tabla de la entidad persistente correspondiente.
  - *get<nombre de la clase>(id)*: obtiene un objeto específico de la base de datos.
  - *getAll<nombre de la clase>()*: obtiene la lista completa de objetos de una tabla específica.
  - *get<nombre de la clase>Count()*: devuelve la cantidad de elementos de una tabla.
  - *update<nombre de la clase>(Object)*: permite actualizar un objeto específico de la base de datos.
  - *delete<nombre de la clase>(Object)*: permite eliminar un objeto específico de la base de datos.
  - *deleteAll<nombre de la clase>()*: permite eliminar todos los objetos de una tabla específica.

  Las funciones *update<nombre de la clase>()* y *delete<nombre de la clase>()* se definen a partir del valor etiquetado *selectable* de un atributo de la entidad persistente.

- ***SQLiteHelper***. Una clase que nos permite administrar tanto la creación de la base de datos y la administración de versiones. También, se definen las tablas de la base de datos a ser utilizadas. Según la estructura de carpetas del código generado, en Android lo encontramos en la carpeta *helpers/*, y en Windows Phone en *Common/*.

El nombre de la base de datos se obtiene del nombre paquete con estereotipo *«DataPersistence»*, el cual contiene al modelo de la persistencia. Siguiendo nuestro ejemplo, el nombre de la base de datos es *"persistencia"* (ver figura 5.7).

- **Definición de las tablas**. Cada tabla generada corresponde a una entidad persistente. Los campos de la tabla corresponden a las propiedades de estas entidades, con el estereotipo *«persistentEntityProperty»*. Los atributos de cada campo se definen mediante los valores etiquetados de cada propiedad de la entidad persistente.

  Por otra parte, en Windows Phone la tabla de base de datos se define y crea directamente a partir del modelo de datos de la entidad correspondiente.

Código generado: Archivos

Por cada clase *«persistentEntity»*, con el valor etiquetado *persistentType* igual a *"File"*, generamos los siguientes elementos:

- **Formulario de datos**. Generamos una pantalla en cada plataforma (en la carpeta *activities/* para Android, y en *Views/* para Windows Phone), donde ponemos a prueba algunas funcionalidades definidas en el *FilesHelper*. En este caso, para el manejo de archivos, utilizamos dos funciones básicas: crear y escribir datos en un archivo, y leer luego dichos datos guardados. El archivo generado se guarda en la carpeta local del dispositivo, en ambas plataformas. El nombre del archivo tiene el siguiente formato: *<nombre de la clase>File.txt*.

- ***FilesHelper***. Conjunto de funciones para el manejo de archivos. Entre las funciones destacadas se encuentran la de crear, leer y escribir un archivo, crear una carpeta, entre otros. Estas funciones están a disposición del usuario.

Código generado: Pares clave-valor

Por cada clase *«persistentEntity»*, con el valor etiquetado *persistentType* igual a *"KeyValue"*, generamos los siguientes elementos:

- **Formulario de datos**. Generamos dos pantallas en cada plataforma, donde ponemos a prueba algunas funcionalidades definidas en el *StorageManager*. Probamos guardar, editar y eliminar datos de estos contenedores.

  En Android (*activities/*), se generan los siguientes archivos:

  - *<nombre de la clase>Activity*: lista los datos almacenados en el *SharedPreferences*. Brinda las opciones para editar y eliminar datos.

  - <*nombre de la clase*>*FormActivity*: formulario que nos permite editar un valor específico del *SharedPreferences*.

  En Windows Phone (*Views/*), se generan los siguientes archivos:

  - <*nombre de la clase*>*View*: lista los datos almacenados en el *LocalStorage*. Brinda las opciones para editar y eliminar datos.
  - <*nombre de la clase*>*DeleteUpdate*: formulario que nos permite editar y eliminar un valor específico del *LocalStorage*.

- ***StorageManager***. Conjunto de funciones para el manejo de los pares clave-valor. En Android hablamos del manejo de los Preferencias Compartidas o *SharedPreferences*. Por otra parte, en Windows Phone hablamos del manejo del *LocalSettings*. Encontramos funcionalidades para el manejo de estos contenedores de datos, tales como guardar datos, obtener un valor específico, limpiar y remover un valor, entre otros. Estas funciones están a disposición del usuario.

Proveedores de datos

A partir del modelo de los proveedores de datos (figura 5.6), utilizamos las siguientes reglas de transformación para generar código para manejar y recibir datos de los servicios REST, los sensores y recibir datos de otras aplicaciones.

Código generado: Interfaces de servicios REST

Tanto en Android como en Windows Phone existen librerías que facilitan el manejo de la conexión a redes. Para permitir la comunicación con servicios REST basados en JSON, optamos por el cliente REST *Retrofit*[10] para Android, y *Base REST service for Universal Apps*[11] para Windows Phone. El código generado es el necesario para la configuración de las librerías en la aplicación, y además la definición de las interfaces de los servicios (*endpoints*) que utilizaríamos con nuestro cliente REST.

En Android, los archivos de configuración de la conectividad se encuentran en la carpeta *conn/*. Se generan dos clases: *ApiCliente.java* y *ApiInterface.java*. En la primera se crea una instancia del cliente REST con *Retrofit*, y se define la URL base del servidor. El nombre de la URL se obtiene de los valores etiquetados de la clase con estereotipo *«WebServiceInterface»* definido: *[protocol]:[domain]*. En *ApiInterface.java*, por otra parte, se definen todos los

[10] *Retrofit*. http://square.github.io/retrofit/
[11] *Base REST service for Universal Apps*. https://github.com/igorkulman/Kulman.WPA81.BaseRestService

endpoints a ser utilizados. Esta lista de servicios se obtiene de la clase con estereotipo *«WebServiceInterface»*, cuyos *operations* tienen el estereotipo *«WebServiceOperation»*.

En Windows Phone, el archivo de configuración de la conectividad se encuentran en la carpeta *Common/*. Se genera una sola clase *ApiInterface.cs*, donde se definen la URL base y los *endpoints* a utilizarse (la generación de código a partir del modelo es la misma que en Android).

De igual forma, generamos unas pantallas para hacer pruebas de los servicios generados. Para esto, generamos *RestInterfaceActivity.java* y *RestInterfaceView* para Android y Windows Phone, respectivamente. En estas clases generamos los esqueletos de las funciones encargadas de trabajar con los servicios (recibir los datos y guardarlos en el modelo respectivo). La idea es que el usuario pueda completar el cuerpo de la función.

Código generado: Sensores

A fin de manejar aquellos proveedores de datos internos de la aplicación móvil, identificamos una serie de elementos correspondiente a sensores y hardware específico del teléfono, que cumplen con esta función. Podemos ver en la figura 5.3 los distintos elementos que identificamos (*HardwareDeviceType*): luz de ambiente, acelerómetro, GPS, giróscopo, brújula, cámara y micrófono. Como veremos, en cada plataforma se siguen caminos diferentes para manejar cada uno de estos elementos.

En Android generamos la pantalla *HardwareDeviceActivity.java* dedicada a poner a prueba los sensores definidos en el modelo. En esta clase *java* agrupamos todos los sensores que mencionamos previamente, pero como dijimos, se manejan de distintas formas:

- *GPS*. Trabajar con este sensor requirió generar una clase que nos permita manejar y obtener datos de este sensor. En *utils/* encontramos la clase *GPSTracker.java*. Para la aplicación *e-market*, se utiliza solamente el GPS.

- *Cámara y micrófono*. Generamos dos pantallas específicas: *CameraActivity* y *AudioActivity*, con las funcionalidades necesarias para manejar y probar la cámara y el micrófono, respectivamente. Ambos sensores almacenan datos en el dispositivo, en forma de archivos, y siguen el siguiente formato de nombres: *"Picture_< currentDate>.jpg"* y *"AudioRecord_<currentDate>.m4a"*, para la foto y el audio respectivamente, donde *currentDate* es la fecha actual en formato texto *"yyyyMMddhhmmss"*.

- Los demás sensores, como el giróscopo, la brújula, el acelerómetro y la luz de ambiente, directamente se instancian y obtienen los datos a partir del manejador de sensores

(*SensorManager*) en la pantalla *HardwareDeviceActivity*. Más detalles sobre el manejo de los sensores en Android podemos encontrarlo en la página del desarrollador [12].

En Windows Phone, de igual forma generamos la pantalla *HardwareDeviceView* con el fin de agrupar los sensores definidos y probarlos. Los sensores los manejamos de forma diferente a Android:

- Cámara y micrófono. Generamos dos pantallas específicas: *CameraView* y *AudioRecordView*, con las funcionalidades necesarias para manejar y probar la cámara y el micrófono, respectivamente. Ambos sensores almacenan datos en el dispositivo, en forma de archivos.

- Los sensores como el giróscopo, la brújula, el acelerómetro, el GPS y la luz de ambiente, se manejan directamente en la pantalla *HardwareDeviceView*. Para la aplicación *e-market*, se utiliza solamente el GPS. Más información sobre el manejo de sensores, lo encontramos en la página del desarrollador de Microsoft[13].

Código generado: Interacción con otras aplicaciones

Una aplicación puede recibir datos de otras aplicaciones. En la figura 5.2 definimos los tipos de datos con los que estaríamos trabajando.

Tanto en Android como en Windows Phone, recibir y manejar estos datos requiere dos pasos: la primera, configurar los permisos de los tipos de datos que recibirá la aplicación; la segunda, recibir el dato y hacer algo con la misma. En Android, los permisos se configuran en el *AndroidManifext.xml* (este código lo generamos automáticamente). Con Windows Phone, la configuración de los permisos se debe hacer manualmente.

Recibir datos de otras aplicaciones se maneja de forma diferente en ambas plataformas. En Android, obtenemos los datos mediante los *intents* (más información ver en la página del desarrollador[14]). Por otra parte, en Windows Phone recibimos los datos con el *ShareTarget* (más información ver el siguiente ejemplo de la página del desarrollador[15]).

[12] *Sensors Overview*. https://developer.android.com/guide/topics/sensors/sensors_overview.html
[13] *Sensors for Windows Phone*. https://msdn.microsoft.com/en-us/library/windows/apps/hh202968(v=vs.105).aspx
[14] *Receiving Simple Data from Other Apps*. https://developer.android.com/training/sharing/receive.html
[15] *Sharing content target app sample* . https://code.msdn.microsoft.com/windowsapps/sharing-content-target-app-e2689782

5.5 Síntesis del capítulo

MoWebA Mobile se enfoca en el desarrollo de aplicaciones móviles nativas centradas en la capa de datos, abarcando los aspectos de persistencia y proveedores de datos. El proceso de desarrollo consiste en utilizar el perfil móvil definido para generar modelos en MagicDraw, exportar estos en formato XMI a Acceleo, y generar código a partir de ellos utilizando las reglas de transformación definidas. El código generado debe ser compilado en los IDEs respectivos del sistema operativo móvil destino, para así obtener la aplicación final.

Para la definición de este ASM móvil, partimos de MoWebA. En primer lugar, extendimos el PIM de entidades de MoWebA a fin de aprovechar elementos conceptuales del tipo estructural. Se extendió la propiedad de una entidad, introduciendo elementos tales como *datatype*, *id*, *size*, entre otros, los cuales enriquecen la definición estructural de este diagrama. En segundo lugar, definimos el metamodelo y perfil UML móvil a partir de las extensiones realizadas. Para la definición de la persistencia, se agregaron elementos como *persistentEntity*, *persistentType* y *persistentEntityProperty*, para poder definir una entidad persistente, establecer un tipo de mecanismo de persistencia para esa entidad y agregar propiedades a los atributos definidos. Para la definición de los proveedores de datos, las interfaces *WebServiceInterface*, *HardwareDeviceInterface* y *MobileAppDataInterface*, permiten definir los proveedores externos, internos y la interoperabilidad con otras aplicaciones. Por último, definimos las reglas de transformación para la obtención automática de los elementos definidos en el modelo. Estas reglas permiten generar aplicaciones completas que permitan hacer pruebas de las funcionalidades definidas. Se genera tanto para Android como para Windows Phone.

Capítulo 6

Evaluación de MoWebA Mobile

A fin de evaluar MoWebA Mobile, en este capítulo se presenta la validación preliminar que preparamos para tal efecto. El propósito de la experiencia de desarrollo es realizar un primer análisis de la propuesta para obtener un primer juicio de intuiciones en cuanto a su usabilidad y portabilidad. Si bien no constituye un experimento formal ni un caso de estudio, es una experiencia que nos permitirá obtener un primer juicio de la propuesta y encaminar a futuros trabajos de validaciones más formales (como experimentos y/o casos de estudio).

Evaluamos la usabilidad y la portabilidad de MoWebA Mobile, con el fin de obtener un mejor entendimiento del mismo. En primer lugar, nos enfocamos en evaluar la usabilidad de nuestro enfoque a fin de obtener primeras valoraciones sobre la experiencia del usuario final, que nos ayuden a realizar mejoras y correcciones, de manera a ir obteniendo un enfoque cada vez más estable, consistente, confiable y fácil de usar. En segundo lugar, en la sección 4.3.3 habíamos mencionado que ciertos aspectos de MoWebA podrían tener impacto positivo en el problema de la portabilidad. Además, señalamos el problema de la fragmentación, el cual incrementa el esfuerzo de desarrollo de aplicaciones móviles, incluyendo la utilización de la persistencia en estos, debido a que las diferentes plataformas móviles manejan de manera distinta los mecanismos de persistencia (sección 3.2). Teniendo en cuenta todo esto, nos interesa evaluar la portabilidad de nuestro enfoque, a fin de obtener unas primeras aproximaciones en pos de una mayor validación de la portabilidad, ya fuera del alcance de este PFC.

En las secciones 6.1.1, 6.1.3 y 6.1.4 se desarrollan y analizan los resultados de la experiencia de validación realizada. En la sección 6.2, se compara el desarrollo manual de una aplicación móvil con el desarrollo utilizando el enfoque propuesto.

6.1 Experiencia de validación

La Organización Internacional de Estándares (ISO 9241-11) define la usabilidad como *"el grado el cual un producto puede ser usado por usuario específicos para lograr objetivos específicos con efectividad, eficiencia y satisfacción en un contexto específico de uso"*. En este contexto, la efectividad se define como la exactitud y la completitud con la que los usuarios pueden alcanzar objetivos específicos; la *eficiencia*, recursos dedicados en relación a la efectividad; y la *satisfacción*, la libertad de disconformidad y actitudes positivas hacia el uso del producto [29]. A fin de evaluar la usabilidad en la experiencia de validación realizada, definimos tareas y utilizamos cuestionarios especializados en la evaluación de dicho aspecto.

Por otra parte, los Requisitos de Calidad y Evaluación de Sistemas y Software, o SQuaRe (ISO/IEC 25010:2011) define la portabilidad como el grado efectividad y eficiencia con el cual un sistema, producto o componente puede ser transferido de un hardware, software u otro entorno operativo a otro [28]. En este PFC nos interesa obtener unas primeras aproximaciones hacia la portabilidad de nuestro enfoque desde el punto de vista de la capa de datos. Comprobaremos en diferentes plataformas la ejecución y el comportamiento de los distintos mecanismos de persistencia definidos.

Para esta experiencia de validación nos guiamos por las actividades que sugieren Genero et al. [19] y Wohling et al. [60] para definir un caso de estudio: diseñar y planificar, preparar la recolección de datos, recolectar los datos, analizar e interpretar los datos recolectados e informar los resultados obtenidos. Si bien, seguimos las directrices de este método de investigación, nuestra experiencia de validación no corresponde un caso de estudio debido a que no es aplicado dentro de un contexto real [19]. Las siguientes secciones desarrollaran cada una de estas actividades.

Los materiales de trabajo utilizados en esta experiencia de validación, así como la corrección y puntuación de los trabajos, estarán disponibles en la página web del proyecto[1].

6.1.1 Diseño

Como primera etapa, nos enfocamos en diseñar los perfiles de los participantes con los que íbamos a trabajar, establecimos los objetivos de la experiencia, identificamos el caso y las unidades de análisis, y por último, definimos las preguntas de investigación que guiarán nuestra experiencia. A continuación hablaremos en más detalle de los puntos mencionados.

Objetivo de la experiencia

[1] Herramientas MoWebA Mobile. http://www.dei.uc.edu.py/proyectos/mddplus/herramientas/mowebamobile/

A fin de definir los objetivos de la experiencia utilizamos el paradigma GQM (*Goal-Question-Metric*), el cual nos permite definirlos en forma operacional y manejable, refinándolos en un conjunto cuantificable de preguntas, que a su vez implica un conjunto de métricas y datos para su recolección [9]. A partir de la plantilla GQM, a continuación definimos los objetivos de nuestra experiencia de validación. Establecemos los perfiles modelador y desarrollador. El primero, sujeto con suficiente conocimiento en el modelado con MoWebA. El segundo, sujeto con experiencia en el desarrollo de aplicaciones móviles, capaz de comprender y hacer modificaciones al código generado.

O1.- **Analizar** el enfoque MDD para el desarrollo de aplicaciones móviles nativas enfocadas en la capa de datos: *MoWebA Mobile*, **con el propósito de** una mejor comprensión **con respecto a** la usabilidad **desde el punto de** vista del modelador **en el contexto de** desarrollo de aplicaciones móviles.

O2.- **Analizar** el enfoque MDD para el desarrollo de aplicaciones móviles nativas enfocadas en la capa de datos: *MoWebA Mobile*, **con el propósito de** una mejor comprensión **con respecto a** la usabilidad y portabilidad **desde el punto de** vista del desarrollador **en el contexto de** desarrollo de aplicaciones móviles.

Caso y unidades de análisis

El caso consiste en el desarrollo de una aplicación móvil para compras *online*, llamada *e-market* (en la sección 5.3 encontramos las especificaciones de esta aplicación).

El desarrollo completo de la aplicación móvil a partir de MoWebA Mobile puede dividirse en las siguientes etapas: modelado con MoWebA Mobile, generación de código con el modelo, generación de la aplicación a partir del código generado y modificaciones a la aplicación.

Siguiendo los diseños y métodos de un caso de estudio de Yin [62], definimos en la figura 6.1 las unidades de análisis, así como el caso y el contexto.

Preguntas de investigación

Teniendo en cuenta lo mencionado, y con el fin de alcanzar los objetivos planteados, se establecen las preguntas de investigación.

- A partir del primer objetivo (O1), tenemos:

O1*PI1* ¿Qué percepción de usabilidad presenta el proceso de modelado del enfoque propuesto?

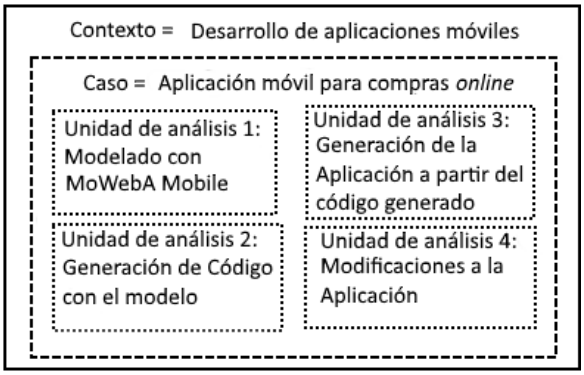


Figura 6.1 Caso único y embebido

O1*PI2* ¿Qué percepción de usabilidad presenta el proceso de generación de código del enfoque propuesto?

O1*PI3* ¿Qué percepción de satisfacción presenta el enfoque MDD propuesto?

- A partir del segundo objetivo (O2), tenemos:

O2*PI1* ¿Qué percepción de usabilidad presenta el proceso de generación de la aplicación a partir del código generado del enfoque propuesto?

O2*PI2* ¿Qué percepción de usabilidad presenta el proceso de realizar una modificación a la aplicación generada?

O2*PI3* ¿Qué percepción de satisfacción presenta el enfoque MDD propuesto?

O2*PI4* ¿Qué percepción de portabilidad presenta el enfoque MDD propuesto?

Perfil de los participantes

A partir de los objetivos establecidos, presentamos los participantes de la experiencia de validación de acuerdo a los perfiles definidos. Creemos que ambos perfiles son lo suficientemente significativos para el tipo de experiencia realizada a fin de lograr nuestras primeras conjeturas en cuanto a la utilidad del enfoque propuesto. De la experiencia de validación participaron:

- *Modeladores*: 6 (seis) alumnos del 8vo semestre de la carrera de Ingeniería Informática de la Universidad Católica "Nuestra Señora de la Asunción". Al momento de la experiencia, estaban culminando la materia Ingeniería del Software I, con suficiente conocimiento en el modelado con MoWebA tras haber desarrollado una aplicación completa con este método y haber sido evaluados por el trabajo realizado en esta materia.

- *Desarrolladores*: 5 (cinco) desarrolladores móviles con experiencia de trabajo para las plataformas Android y/o Windows Phone. Específicamente, 4/5 contaban con experiencia en desarrollo de aplicaciones móviles para Android, y 1/5 para Windows Phone. La experiencia promedio de desarrollo va de 1 a 1.5 años en la industria.

6.1.2 Planificación

La experiencia de validación fue dividida en las siguientes etapas: proceso de modelado, proceso de generación de código, proceso de generación de la aplicación a partir del código generado y proceso de modificación de la aplicación. El modelador se encargó del proceso de modelado y de la generación de código. El desarrollador se ocupó de la generación de la aplicación a partir del código generado, y de realizar modificaciones a la aplicación generada.

Durante la experiencia de validación se realizó el modelado y la generación de código de la aplicación móvil *e-market* para compras *online* (las especificaciones están definidas en la sección 5.3). Partiendo de MoWebA Mobile, se requirieron los modelos de persistencia y de proveedores, a partir de los cuales se generó código nativo para Android y Windows Phone. Para generar la aplicación móvil fue necesario importar el código generado a los respectivos IDEs a fin de compilarlos. Por último, la modificación realizada a la aplicación consistió en cambiar el tipo de mecanismo de persistencia de una entidad (un grupo realizó modificaciones al código, y otro, al modelo).

A continuación detallamos el paso a paso de las sesiones de trabajo realizadas durante la experiencia de validación; y además, hablamos de los cuestionarios seleccionados para ser utilizados durante la experiencia.

Cuestionarios utilizados

Utilizamos dos cuestionarios para medir la usabilidad del enfoque propuesto, *After Scenario Questionnaire* (ASQ) y *System Usability Scale* (SUS). Por otra parte, para obtener unas primeras aproximaciones sobre la portabilidad del enfoque propuesto, utilizamos un cuestionario de preguntas abiertas en cuestión. A continuación describimos cada uno ellos.

Al desarrollar sistemas computacionales, es importante implementar un método para medir la satisfacción del usuario con los sistemas existentes o prototipos de futuros sistemas. A fin de evaluar la satisfacción del usuario durante la participación en los estudios de usabilidad basados en escenarios, encontramos los cuestionarios ASQ. Este cuestionario es administrado después de cada escenario, y parece ser lo suficientemente sensible como para ser una medida útil para la usabilidad [37]. Se enfoca en la medición de tres aspectos principales de satisfacción: la facilidad de completar la tarea, satisfacción con el tiempo de finalización y satisfacción con la información de soporte [6]. Como vemos en la figura 6.2,

consta de tres ítems, con una de escala calificación de 7 puntos que va desde el fuertemente desacuerdo al fuertemente acuerdo. La media aritmética de los tres ítems constituye el puntaje general [3]. Con respecto a este puntaje, apuntamos que valores más cercanos al 1, en una escala de 7 puntos, indican mayor nivel de satisfacción.

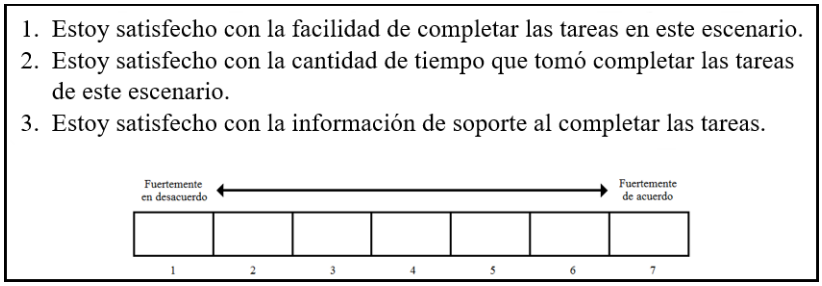


Figura 6.2 Cuestionario ASQ

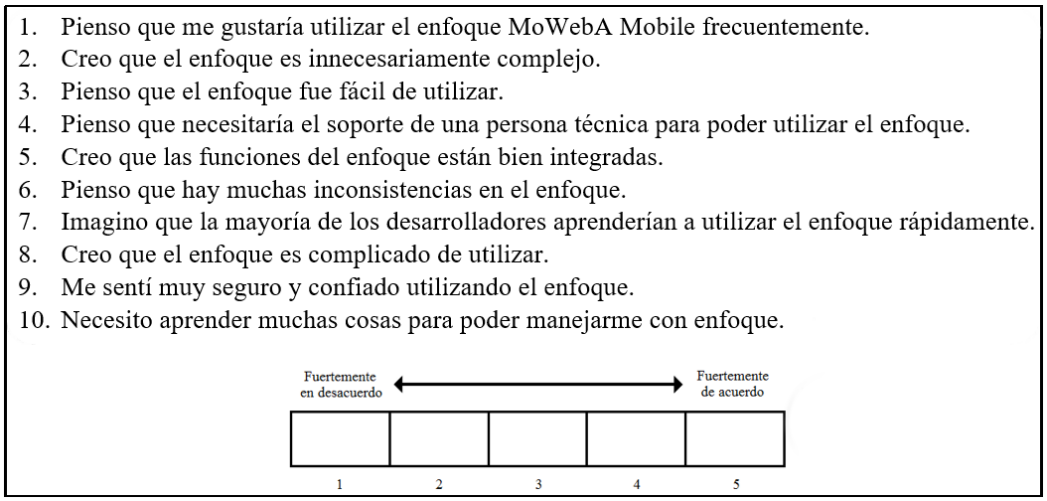
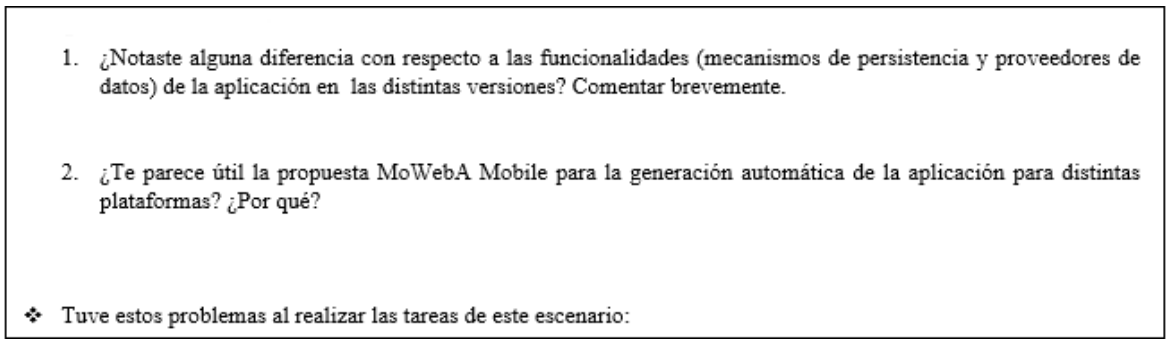


Figura 6.3 Cuestionario SUS

De acuerdo a Brooke et al. [13], SUS proporciona una visión global de las evaluaciones subjetivas de usabilidad. Generalmente se utiliza después de que el encuestado haya tenido la oportunidad de utilizar el sistema que se está evaluando, pero antes de que se realice cualquier debate o discusión. SUS ha demostrado ser una herramienta valiosa de evaluación, siendo robusto y confiable. Se enfoca principalmente en la medición de la satisfacción del usuario (eficiencia, capacidad de aprendizaje y satisfacción), cubriendo una variedad de aspectos de usabilidad del sistema, como la necesidad de soporte, capacitación, y complejidad [6]. Como vemos en la figura 6.3, es una simple escala de diez ítems, cada una con una escala de 5 puntos (va de fuertemente desacuerdo a fuertemente acuerdo). La contribución del puntaje de los ítems con tono positivo (los impares), es la posición de la escala menos 1. Los ítems

con tono negativo (los pares), tienen la contribución de puntaje de 5 menos la posición de la escala. Luego, para obtener el valor total del SUS se multiplica la suma de los puntajes por 2.5. Los puntajes del SUS tiene un rango de 0 a 100. Para tener una mejor idea de cómo usar esta información, Sauro y Lewis [54] proponen transformar el puntaje obtenido a un rango percentil. Tras analizar 446 estudios y más de 5000 respuestas individuales del SUS, han encontrado que cualquier resultado con un rango percentil inferior al 50% está, por definición, por debajo del promedio; y cualquiera por encima del 50%, se encuentra por encima del promedio. Este rango percentil dice qué tan usable es la aplicación relativo a otros productos analizados por Sauro (estudios y resultados analizados de cuestionarios SUS, presentes en una base de datos).

Por último, a fin de obtener las primeras aproximaciones sobre la portabilidad de nuestro enfoque, preparamos un cuestionario con preguntas abiertas (ver figura 6.4) enfocado en obtener la percepción del usuario respecto a la portabilidad del enfoque. Se busca conocer: i) la existencia de diferencias entre los mecanismos de persistencia generados para las plataformas Android y Windows Phone; y ii) la utilidad del enfoque para la generación automática de aplicaciones móviles para distintas plataformas.



1. ¿Notaste alguna diferencia con respecto a las funcionalidades (mecanismos de persistencia y proveedores de datos) de la aplicación en las distintas versiones? Comentar brevemente.

2. ¿Te parece útil la propuesta MoWebA Mobile para la generación automática de la aplicación para distintas plataformas? ¿Por qué?

❖ Tuve estos problemas al realizar las tareas de este escenario:

Figura 6.4 Cuestionario - Primeros pasos hacia la portabilidad

Procedimiento de la experiencia

La experiencia de validación fue realizada en tres sesiones de trabajo. El procedimiento de trabajo llevado a cabo en cada sesión se describe a continuación. En estas sesiones se realizaron diferentes tareas o escenarios (ver cuadro 6.2).

Sesión 1: Exposición del enfoque

- El investigador realizó una exposición del enfoque propuesto (MoWebA Mobile) a los participantes de la experiencia.
- A cada participante se le entregó los materiales de trabajo, descritos en el cuadro 6.1.

Sesión#	Actividad realizada	Cantidad de participantes	Perfil	Duración (minutos)	Herramientas utilizadas	Materiales de trabajo
1	Explicación del enfoque y desarrollo de la aplicación ejemplo ***MovistarWops***	7	Modelador	150	MagicDraw y Acceleo	Reglas de transformación, Perfil MoWebA Mobile y modelo de clase de la aplicación ***MovistarWops***
2	Modelado y generación de código de la aplicación *e-market*	6	Modelador	140	MagicDraw y Acceleo	Reglas de transformación, perfil MoWebA Mobile y modelo de clase de la aplicación, manual de la propuesta MoWebA Mobile y guía de ejercicios
3	Generación de la aplicación *e-market* a partir del código generado	5	Desarrollador	203	MagicDraw, Acceleo, Android Studio y Visual Studio	Reglas de transformación y el código generado en S1, perfil MoWebA Mobile y modelo de persistencia de la aplicación *e-market* (ver figura 5.5), manual de la propuesta MoWebA Mobile y manuales con información útil para trabajar con Android y Visual Studio

Cuadro 6.1 Descripción de cada una de las sesiones de trabajo llevadas a cabo en el marco de la experiencia de validación

Escenarios	Tareas comprendidas
Modelado (M1)	Diseñar: entidades persistentes, servicios, sensores y la interoperabilidad con otras aplicaciones.
Generación de código (GC1)	Exportar el modelo desde MagicDraw e importarlo en Acceleo, ejecutar el generador de código.
Generación de la aplicación a partir del código generado (GP1)	Importar el código generado al IDE (Android Studio y/o Visual Studio), realizar los ajustes necesarios previo a la generación (importar librerías) y ejecutar la compilación.
Portabilidad (P1)	Verificar los mecanismos de persistencia (CRUD de datos) y las funcionalidades presentadas (utilidades y clases auxiliares).
Modificaciones a la aplicación generada: manual (MD1)	Modificar manualmente el código generado, generar la aplicación y verificar los cambios realizados.
Modificaciones a la aplicación generada: modelado (MD2)	Modificar los modelos de la aplicación, exportarlos desde MagicDraw, importarlos en Acceleo, ejecutar el generador de código, generar la aplicación a partir del código generado y verificar los cambios realizados.

Cuadro 6.2 Lista de escenarios utilizados en la experiencia de validación.

- El investigador y los participantes realizaron un ejemplo sencillo de modelado y generación de código de la aplicación *MovistarWops*[2].
- El investigador se encargó de generar la aplicación móvil a partir del código generado del ejemplo realizado.

Sesión 2: Desarrollo de la aplicación móvil - modeladores

[2]La aplicación llamada *MovistarWops* tiene como función registrar las instalaciones de antenas satelitales, realizadas por una compañía telefónica. La misma debe funcionar sin conexión a red y poder registrar datos relacionados a la instalación en el teléfono para sincronizarlos luego.

- El investigador presentó a los participantes la aplicación a desarrollar (aplicación *e-market*) y proveyó las indicaciones del trabajo.
- A cada participante se le entregó los materiales de trabajo, descritos en el cuadro 6.1.
- Los participantes, sin intervención del investigador, elaboraron el modelo de la aplicación. En esta etapa, los participantes debieron avanzar en el modelado todo lo que podían, sin realizar consultas al investigador. El investigador y su colaborador tomaron el tiempo de este proceso.
- Los participantes, con ayuda del investigador, realizaron correcciones al modelo. En esta etapa, los participantes consultaron al investigador todas aquellas dudas respecto al modelado, a fin de alcanzar el modelado completo de la aplicación. El investigador solicitó que las modificaciones realizadas fueran guardadas en una carpeta diferente. El investigador y su colaborador tomaron el tiempo de este proceso.
- Los participantes completaron un cuestionario ASQ sobre el proceso de modelado (escenario M1).
- Los participantes generaron código a partir del modelo elaborado. El investigador y su colaborador tomaron el tiempo de este proceso.
- Los participantes completaron un cuestionario ASQ sobre el proceso de generación de código (escenario GC1).
- Los participantes completaron un cuestionario SUS sobre el enfoque completo utilizado.

Sesión 3: Desarrollo de la aplicación móvil - desarrolladores

- El investigador realizó una breve exposición del enfoque propuesto (MoWebA Mobile) a los participantes de la experiencia. Se presentó la aplicación a desarrollar (aplicación *e-market*) y se proveyó las indicaciones del trabajo.
- A cada participante se le entregó los materiales de trabajo, descritos en el cuadro 6.1.
- El investigador brindó indicaciones de cómo importar el código generado a los respectivos IDEs.
- Los participantes importaron el código generado y generaron la aplicación. El investigador y su colaborador tomaron el tiempo de este proceso.

- Los participantes completaron un cuestionario ASQ sobre el proceso de generación de la aplicación a partir del código generado (escenario GP1).
- Los participantes realizaron verificaciones de las funcionalidades generadas de la aplicación en los distintos teléfonos móviles. Todos los participantes experimentaron la aplicación generada en ambas plataformas, Android y Windows Phone (escenario P1).
- Los participantes completaron un cuestionario de portabilidad (escenario P1).
- Los participantes realizaron modificaciones a la aplicación generada. El investigador brindó las indicaciones sobre las modificaciones a realizar. Luego, se categorizó a los participantes en dos grupos:
  - El primer grupo realizó modificaciones manuales, es decir, modificaciones directas al código de la aplicación generada. 3/5 participantes desarrollaron manualmente para Android. Las modificaciones realizadas realizadas fueron probadas en los teléfonos móviles, a fin de verificar los cambios realizados.
  - El segundo grupo realizó modificaciones al modelo de la aplicación. Entre los desarrolladores, 2/5 participantes poseían experiencia en el desarrollo dirigido por modelos, a los cuales se les otorgó los modelos de la aplicación (utilizados para generar la aplicación *e-market*). Estos debían seguir todo el proceso de desarrollo con el enfoque MoWebA Mobile, hasta la ejecución de la aplicación contemplando los cambios solicitados. En este grupo, una persona generó para Android y otra para Windows Phone.

  Se tomó el tiempo de este proceso de modificación. Además, se les indicó a los participantes que guarden las modificaciones realizadas en carpetas diferentes.
- Los participantes completaron un cuestionario ASQ sobre el proceso de modificación de la aplicación generada (escenarios MD1 y MD2).
- Los participantes completaron un cuestionario SUS sobre el enfoque completo utilizado.

En la sesión 3, los cuestionarios SUS y ASQ se complementaron con preguntas abiertas, con la intención de recoger toda la información posible de los participantes. En los ASQ, se sugirió comentar sobre los posibles problemas al realizar las tareas del respectivo escenario. En el SUS, por otra parte, la intención fue conocer la opinión de los participantes respecto a que si los requerimientos iniciales fueron cubiertos o no por la aplicación generada, y las sugerencia y/o comentarios que tenían sobre el enfoque en general.

6.1.3 Preparación y recolección de datos

De la experiencia de validación se consiguieron datos cuantitativos (con los cuales responder a las preguntas de investigación), y datos cualitativos (basados en opiniones personales) para complementar los primeros y mejorar la percepción global obtenida.

Para la recolección de los datos utilizamos tres fuentes de información: A) la documentación del proyecto (modelo sin correcciones, modelo con correcciones, código generado, código de la aplicación importada al IDE con ajustes manuales y código de las modificaciones de la aplicación), B) la planilla de medición de los tiempos de las sesiones de trabajo, y C) los cuestionarios. Los *datos cuantitativos* se obtuvieron de la documentación del proyecto, de la planilla de medición de tiempos y los cuestionarios. Por otra parte, los *datos cualitativos* se obtuvieron de las opiniones y/o comentarios de los participantes (agregados a los cuestionarios utilizados). A continuación hablaremos de cada fuente de información utilizada y qué datos hemos obtenido a partir de ellos.

Fuentes de información

A) **Documentación del proyecto.**

- De los modeladores obtuvimos los modelos elaborados sin intervención del investigador, los modelos elaborados con ayuda, y el código generado automáticamente a partir del modelo.

  A partir de estos documentos, por cada modelador se obtuvo la cantidad de correcciones de modelado, correspondiente a la cantidad de elementos de modelado por agregar, modificar o eliminar para obtener un modelo completo.

  Con los datos recabados fue posible calcular las tasas de éxito para el proceso de modelado y para el proceso de generación de código. Posteriormente, a fin de tener una visión general del rendimiento de los modeladores, se tomaron todos los datos y se calcularon los siguientes promedios: la tasa de éxito promedio para el proceso de modelado y la tasa de éxito promedio para el proceso de generación de código.
- De los desarrolladores obtuvimos el código de la aplicación ya importado en los respectivos IDEs, y el código de la aplicación con las modificaciones solicitadas. A partir de estos documentos, por cada desarrollador se obtuvieron los siguientes datos:
  - La cantidad de correcciones al código de la aplicación, correspondiente a la cantidad de elementos por agregar, modificar o eliminar para obtener una aplicación completa.

- La cantidad de líneas de código generadas automáticamente (importadas al IDE correspondiente).
- La cantidad de líneas de código agregadas luego de la modificación solicitada.

Con los datos recabados fue posible calcular las tasas de éxito para el proceso de generación de la aplicación a partir del código generado, y para el proceso de modificaciones sobre la aplicación generada. Posteriormente, a fin de tener una visión general del rendimiento de los desarrolladores, se tomaron todos los datos y se calcularon los siguientes promedios: la tasa de éxito promedio para el proceso de generación de la aplicación a partir del código generado y para las modificaciones realizadas.

B) **Planillas de medición de tiempos**.

Las planillas de medición de tiempos nos permitieron registrar la duración de cada una de las actividades realizadas durante el desarrollo la experiencia de validación.

De los modeladores se registraron la duración del modelado sin intervención del investigador, el tiempo de modelado con ayuda del investigador y el tiempo de generación automática de código. Por otra parte, de los desarrolladores se registraron el tiempo de la generación de la aplicación a partir del código generado, y la duración del desarrollo de las modificaciones solicitadas.

A partir de los tiempos registrados en las planillas, se obtuvo el tiempo promedio de cada actividad y el promedio del tiempo de desarrollo total de la aplicación utilizando el enfoque propuesto.

C) **Cuestionarios**.

Con los cuestionarios pudimos recabar datos cualitativos (por medio de comentarios y opiniones personales) y cuantitativos (puntaje obtenido del cuestionario).

El análisis cuantitativo de los cuestionarios se realizó de la siguiente manera: por cada modelador y desarrollador se analizaron los cuestionarios ASQ, obteniendo el puntaje correspondiente a la actividad realizada. Posteriormente, se obtuvo el promedio de puntajes por actividad. De igual forma, por cada modelador y desarrollador se obtuvieron los puntajes del cuestionario SUS. A partir de esto, se pudo obtener el promedio correspondiente por cada perfil mencionado y el promedio general de la experiencia.

El análisis cualitativo nos permitió recopilar las dificultades y problemas que se presentaron en cada actividad realizada por los desarrolladores (complemento al

cuestionario ASQ), así como sus opiniones generales sobre el enfoque propuesto: correctitud de la aplicación generada teniendo en cuenta los requerimientos iniciales, y sugerencias y/o comentarios sobre su experiencia de desarrollo con MoWebA Mobile. De igual forma, obtuvimos comentarios sobre la portabilidad del enfoque, analizando la percepción de los participantes sobre la existencia de diferencias entre los mecanismos de persistencia generados para las plataformas Android y Windows Phone, y sus opiniones sobre la utilidad del enfoque para la generación automática de aplicaciones móviles para distintas plataformas.

Cómo medimos la usabilidad

ISO 9241-11[29] sugiere que las medidas de usabilidad deben cubrir *efectividad* (la habilidad de los usuarios de completar tareas usando el sistema, y la calidad de resultado de esas tareas), *eficiencia* (el nivel de recursos consumidos en la realización de las tareas) y *satisfacción* (las reacciones subjetivas de los usuarios al uso del sistema). Las medidas precisas dentro de cada una de estos aspectos puede variar ampliamente [13].

A continuación, explicamos cómo se van a medir los distintos aspectos de la usabilidad:

- En el proceso de modelado:
  - *Efectividad*: se mide la tasa de éxito promedio del proceso de modelado. Cada tarea tuvo un proceso de corrección, basados en una escala de puntuación[3] se logró obtener un puntaje de completitud o éxito de la tarea.
  - *Eficiencia*: se mide el tiempo promedio de finalización del modelado, sin intervención y con intervención del investigador. El tiempo total del modelado de obtiene de la suma promedio de ambas etapas del modelado.
  - *Satisfacción*: se mide a través del promedio del ASQ para el modelado.
- En el proceso de generación de código:
  - *Efectividad*: se mide la tasa de éxito promedio de la generación de código. La corrección se realizó de igual forma que en el proceso de modelado, pero con la escala de puntuación adaptada a esta tarea.
  - *Eficiencia*: se mide el tiempo promedio de finalización del proceso de generación automática de código.
  - *Satisfacción*: se mide a través del promedio del ASQ para la generación de código.

[3]Para cada tarea se estableció una lista de indicadores que significaban la completitud de dicha tarea; luego, se establecieron puntajes a cada indicador (100 puntos distribuidos).

- En el proceso de generación de la aplicación:
  - *Efectividad*: se mide la tasa de éxito promedio de la generación de la aplicación. La corrección se realizó de igual forma que en el proceso de modelado, pero con la escala de puntuación adaptada a esta tarea.
  - *Eficiencia*: se mide el tiempo promedio de finalización del proceso de generación de la aplicación a partir del código generado.
  - *Satisfacción*: se mide a través del promedio del ASQ para la generación de la aplicación.
- En el proceso de modificación de la aplicación generada:
  - *Efectividad*: se mide la tasa de éxito promedio de la modificación realizada. La corrección se realizó de igual forma que en el proceso de modelado, pero con la escala de puntuación adaptada a esta tarea.
  - *Eficiencia*: se mide el tiempo promedio de finalización del proceso de modificación.
  - *Satisfacción*: se mide a través del promedio del ASQ para la modificación de la aplicación.
- Para el enfoque MDD en general:
  - *Satisfacción*: se mide a través del promedio y la desviación estándar del puntaje obtenido en el SUS.

Cómo experimentamos la portabilidad

Muñoz et al. [43] destacan que las métricas definidas en ISO 9126 no aportan a la evaluación de la portabilidad en el enfoque MDD, porque están enfocadas en re-usar el software desarrollado. En contraste, los enfoques MDD permiten ir un paso atrás y re-compilar el modelo conceptual a diferentes plataformas tecnológicas usando diferente PSMs y compiladores (asegurando portabilidad). Teniendo en cuenta esto, nos encontramos con limitaciones en búsqueda de alternativas y métodos de evaluación de la portabilidad de nuestro enfoque. Optamos por obtener unas primeras aproximaciones hacia una evaluación más formal de esta calidad del software, realizando pruebas con la aplicación generada y recolectando la percepción de los participantes al respecto.

Durante la sesión 3, los desarrolladores verificaron y analizaron el código generado, específicamente las funcionalidades generadas y los distintos mecanismos de persistencia adaptadas para cada plataforma móvil. Luego de esta verificación, se realizaron pruebas con

la aplicación generada en distintas plataformas móviles: Android 6.0.1 (teléfonos Nexus 5 y Samsung S5), Android 4.4.2 (teléfono Samsung Galaxy Note 2), Windows Phone 8.1 (teléfono Windows Phone) y Windows 10 Mobile (emulador de Windows Phone en Visual Studio). Las pruebas consistían en cargar datos e ir observando el almacenamiento por medio de los distintos mecanismos. Además, se verificaron los proveedores de datos definidos (sensores, cámara, entre otros). Esto nos permitió recolectar las opiniones de los participantes sobre esta calidad del software en nuestro enfoque.

A partir de las pruebas realizadas, se pudo recabar las opiniones y comentarios de los desarrolladores sobre:

- las posibles diferencias en las funcionalidades y mecanismos de persistencia, en las distintas plataformas;
- la utilidad del enfoque para la generación automática de aplicaciones móviles para distintas plataformas.

Amenazas a la validez de la experiencia

A continuación se comentan algunos aspectos que pueden haber atentado contra la validez del experimento y cómo se intentaron mitigar[19]:

- Validez interna: *definido como el grado de confianza en una relación causa-efecto entre los factores de interés y los resultados observados.* Teniendo en cuenta este aspecto, se formaron dos grupos bien diferenciados balanceando el nivel y área de experiencia: por una parte los modeladores (alumnos con conocimiento suficiente en el modelado con MoWebA); y por otra parte, los desarrolladores móviles (con experiencia promedio de 1 a 1.5 años en la industria). Para evitar el ausentismo de los alumnos durante las sesiones, la experiencia se realizó en horario de clase de la materia Ingeniería del Software I, indicando a los alumnos que las tareas a realizarse podrían aparecer en el examen. Respecto a los desarrolladores, se estuvo en constante comunicación con ellos, organizando un horario en que todos puedan asistir. Por último, cada participante trabajó en forma individual, en máquinas diferentes; y además, se supervisaba que los participantes no se comunicaran entre sí, evitando así posibles plagios en la experiencia.
- Validez externa: *representa el grado hasta el que los resultados alcanzados pueden generalizarse.* Este aspecto pudo verse afectado por la cantidad de participantes involucrados (6 alumnos y 5 desarrolladores móviles). Si bien, este número no es significativo para fines estadísticos, es apropiado para poder emitir primeras

aproximaciones y juicios iniciales, en pos de realizar validaciones más formales, como experimentos, casos de estudio y/o experiencias con mayor cantidad de participantes, que nos permitan obtener conclusiones significativas y precisas. Además, el caso de desarrollo fue simple, pero no muy alejado de los requerimientos que un caso de la industria podría implicar. Este caso contempló el desarrollo de una aplicación móvil teniendo en cuenta todos los aspectos cubiertos por nuestro enfoque.

- Validez de constructo: *refleja hasta qué punto las medidas que se han realizado se adecuan a lo que el investigador tiene en mente y a lo que se está investigando.* Las medidas seleccionadas son utilizadas normalmente para medir los aspectos de calidad del software. En específico, para medir la usabilidad se utilizaron cuestionarios estándares, considerados confiables y válidos: SUS [6] y ASQ [37]. Por otra parte, la medición de la portabilidad pudo haberse afectado por no contar con cuestionarios estándares y/o métricas válidas, pero consideramos adecuado para emitir primeras aproximaciones en pos de realizar validaciones más formales.

- Fiabilidad: *indica la dependencia de los datos y su análisis respecto de un investigador específico y la capacidad de replicar el mismo estudio y obtener los mismos resultados.* La cadena de evidencias desde todas las fuentes de información involucradas en la experiencia, se llevó a cabo respetando la literalidad de los datos obtenidos para evitar introducir sesgos a través de la interpretación.

6.1.4 Análisis e interpretación de resultados

- O1*PI1: ¿Qué percepción de usabilidad presenta el proceso de modelado del enfoque propuesto?*

  Como vimos, el proceso de modelado se dividió en dos etapas: i) el modelado sin intervención alguna del investigador (SII, Sin Intervención del Investigador), y ii) el modelado con ayuda del investigador (CII, Con Intervención del Investigador). En el cuadro 6.3 resumimos los resultados recabados durante esta etapa. A continuación brindamos detalles de estos resultados.

  En la primera etapa, los modeladores realizaron el modelado de la aplicación sin la posibilidad de hacer preguntas al investigador. Este período de trabajo tuvo una duración promedio 39.17 minutos, con una tasa de éxito promedio del 29%. Si bien, este número no representa un avance significativo en el desarrollo de la aplicación

solicitada, corresponde a la primera experiencia de trabajo con el enfoque por parte de los modeladores.

Todos los participantes trabajaron sobre el modelo de persistencia, de los cuales el 33% tuvo errores utilizando el valor etiquetado *persistentType*. El 50% alcanzó a trabajar sobre el modelo proveedores, pero con errores en el modelado con estereotipos y valores etiquetados.

En la segunda etapa, los modeladores consultaron al investigador las dudas que tenían. Cabe resaltar que 2/6 modeladores no necesitaron ayuda alguna y pudieron finalizar exitosamente esta etapa. La duración promedio de este segundo periodo de modelado fue de 26 minutos, alcanzando una tasa de éxito promedio del 82%.

En total, el modelado de la aplicación tuvo una duración promedio de 64.17 minutos, alcanzando una tasa de éxito promedio del 82%. El puntaje ASQ promedio obtenido fue de 2.89. El puntaje obtenido refleja un nivel de satisfacción bueno por parte de los modeladores con el proceso de modelado.

Algunos errores comunes encontrados durante las correcciones llevadas a cabo en los modelos fueron: el modelado de los servicios (en el modelo de proveedores) incompleto o con algunos errores; y, la correcta estructuración de las carpetas del proyecto (algunos olvidaron colocar cada modelo separados en carpetas específicas).

Escenario	Tasa de Éxito Promedio SII	Tiempo de Finalización Promedio SII (minutos)	Tasa de Éxito Promedio CII	Tiempo de Finalización Promedio CII (minutos)	Tiempo Total de Finalización Promedio (minutos)	Puntaje ASQ Promedio
Modelado	29%	38.17.	82%	26	64.17	2.89

Cuadro 6.3 Mediciones de usabilidad del proceso de modelado

- O1*PI2: ¿Qué percepción de usabilidad presenta el proceso de generación de código del enfoque propuesto?*

  En el cuadro 6.4 resumimos los resultados recabados durante esta etapa. A continuación brindamos detalles de estos resultados.

  Si bien, todos los participantes pudieron completar satisfactoriamente las tareas de este escenario, la duración promedio de este escenario fue de 15.33 minutos. Este número es significativamente alto para una generación automática. Esto se debió principalmente a que 2/6 participantes presentaron errores durante la generación del código (se gastó tiempo en la detección de dichos errores). Estos errores correspondían a imperfecciones encontradas en el modelo, durante el modelado con la herramienta MagicDraw; específicamente, al borrar elementos del diagrama, la herramienta no

los eliminaba por completo del proyecto, generando así imperfecciones al momento de exportar el modelo. Pero, a pesar de los errores mencionados, la aplicación pudo generarse exitosamente.

El puntaje promedio obtenido del ASQ fue de 1.94. Este puntaje refleja un nivel de satisfacción muy bueno por parte de los modeladores en el proceso de generación de código.

Escenario	Tasa de Éxito Promedio	Tiempo de Finalización Promedio (minutos)	Puntaje ASQ Promedio
Generación de Código (GC1)	100%	15.33	1.94

Cuadro 6.4 Mediciones de usabilidad del proceso de generación de código

- O1*PI3: ¿Qué percepción de satisfacción presenta el enfoque MDD propuesto?*

  Al hablar del cuestionario SUS (sección 6.1.2), mencionamos que la mejor forma de interpretar los resultados es normalizando y obteniendo el rango percentil. Cualquier resultado con un rango percentil inferior al 50 % está, por definición, por debajo del promedio, y cualquiera por encima del 50 % está por encima del promedio.

  La percepción de satisfacción de MoWebA Mobile por parte de los modeladores tuvo un puntaje promedio de 50 en el SUS, con una desviación estándar del 16.89. Convirtiendo este puntaje al rango percentil de Sauro, obtenemos un valor de 13%. Este resultado está por debajo del promedio.

  Analizando los cuestionarios SUS, creemos que este puntaje se debe a la percepción de complejidad de los modeladores respecto al enfoque, algo inseguros en la utilización de la misma y con la necesidad de aprender muchas cosas para poder manejar el enfoque. Esto pudo deberse a la primera experiencia de trabajo que tuvieron con MoWebA Mobile, dándonos indicios de que para minimizar la curva de aprendizaje podrían requerirse más clases de preparación. Esto nos llama la atención puesto que se brindaron materiales informativos, e incluso el proceso de modelado de la aplicación no se realizó desde cero, sino que partió del esqueleto de la aplicación (modelo de clase), minimizando así la complejidad del modelado (se buscaba que todos los alumnos lleguen al modelado completo, para así poder generar el código).

- O2*PI1: ¿Qué percepción de usabilidad presenta el proceso de generación de la aplicación a partir del código generado del enfoque propuesto?*

  El código generado de la aplicación partió de un modelo completo. A fin de tener una aplicación generada unificada para todos, no se utilizaron los modelos diseñados

por los modeladores. Como vimos en O1*PI1*, ninguno había alcanzado el 100% de completitud durante ese escenario.

En el cuadro 6.5 resumimos los resultados recabados durante esta etapa. A continuación brindamos detalles de estos resultados.

Esta etapa tuvo una tasa de éxito del 98%, con una duración de 51.6 minutos. El tiempo de finalización promedio mencionado es alta, debido a inconvenientes que surgieron al momento de importar el código generado en Acceleo a los respectivos IDEs. Si bien, todos los participantes pudieron generar correctamente la aplicación, muchos tuvieron dificultades con el manejo de los IDEs y con el proceso de creación del proyecto a partir del código generado. Teniendo en cuenta lo primero, las dificultades técnicas pudieron haberse evitado con una mejor preparación pre-experiencia. Pero lo segundo nos llama la atención, a pesar de la experiencia de trabajo que poseen los desarrolladores, crear el proyecto y posterior re-estructuración a partir del código generado, llevó más tiempo de lo esperado. A partir de estos resultados, nos queda analizar vías que nos permitan mejorar la realización de este proceso: una mejor estructuración del proyecto, o inclusive una automatización completa de este proceso.

El promedio de líneas de código (o LoC, *Lines of Code*) generadas fueron: para Android, 6411; y para Windows Phone, 3828. Hay que tener en cuenta que esta cantidad de líneas se obtuvo contabilizando el proyecto creado por los desarrolladores, por lo que hubo ciertas variaciones en cada caso. Las líneas de código contabilizadas incluyen líneas en blanco, comentarios y los *imports* en cada clase.

El puntaje promedio obtenido del ASQ fue de 2.20. Este puntaje refleja un nivel de satisfacción bueno por parte de los desarrolladores durante el proceso de generación de la aplicación a partir del código generado.

Escenario	Tasa de Éxito Promedio	Tiempo de Finalización Promedio (minutos)	Líneas de código (Android)	Líneas de código (Windows Phone)	Puntaje ASQ Promedio
Generación de la aplicación a partir del código generado (GP1)	98%	51.6	6411	3828	2.20

Cuadro 6.5 Mediciones de usabilidad del proceso generación de la aplicación a partir del código generado

Comentarios recolectados. Durante esta actividad, los principales problemas destacados y mencionados por los desarrolladores se debieron al uso del IDE. Por una parte, hubo problemas con la configuración del IDE, desperdiciándose tiempo en la instalación de actualizaciones requeridas e importación de librerías. Por otra parte,

problemas en la importación del código generado, requería crear el proyecto en el IDE y luego ir ubicando el código en las respectivas carpetas.

- O2*PI2: ¿Qué percepción de usabilidad presenta el proceso de realizar una modificación a la aplicación generada?*

  Las modificaciones realizadas fueron: (i) modificación manual (modificando directamente el código generado); y, (ii) modificación al modelo (utilizando el enfoque MoWebA Mobile). Con esta actividad, buscamos comparar el esfuerzo y la dificultad de realizar una modificación a la aplicación generada con ambos enfoques de desarrollo: manual y automático. Los cuadros 6.6 y 6.7 presentan los resultados recabados durante esta etapa. A continuación brindamos detalles de estos resultados.

  La modificación realizada consistió en cambiar el tipo de mecanismo de persistencia de una entidad: pasar de pares clave-valor a base de datos. Se trabajó en dos grupos, por una parte el desarrollo manual involucró tres participantes (todos desarrollaron para Android); y por otra parte, el desarrollo con el enfoque MoWebA Mobile involucró dos participantes (uno desarrollando para Android y otro para Windows Phone). De todos los participantes, solo el 40% pudo completar en totalidad las modificaciones solicitadas. La tasa de éxito promedio obtenido fue del 72%.

  Del primer grupo, con las modificaciones manuales (cuadro 6.6), se obtuvo una tasa de éxito promedio del 73.3%. Solamente 1/3 pudo finalizar en totalidad la modificación solicitada. Los otros dos participantes alcanzaron el 50% del desarrollo, que por falta de tiempo, no alcanzaron mostrar los datos en pantalla. El promedio de tiempo de finalización fue de 57 minutos, y el promedio del puntaje ASQ obtenido en este grupo fue de 3. Este puntaje refleja un nivel de satisfacción bueno por parte de los desarrolladores durante la modificación manual de la aplicación. Este puntaje es más alto en comparación con los demás puntajes obtenidos, y refleja la inconformidad de los participantes con la dificultad para realizar la tarea, y con el tiempo disponible para realizarla.

Escenario	Tasa de Éxito Promedio	Tiempo de Finalización Promedio (minutos)	Puntaje ASQ Promedio
Modificaciones (MD1)	73.3%	57	3

Cuadro 6.6 Mediciones de usabilidad sobre el proceso de realizar una modificación a la aplicación generada: modificaciones manuales

En el segundo grupo, las modificaciones realizadas utilizando MoWebA Mobile (cuadro 6.7), se obtuvo una tasa de éxito promedio del 72%. De los participantes,

solamente uno pudo culminar la modificación solicitada y probar con éxito los cambios realizados (aplicación para Windows Phone). El otro participante (aplicación para Android), pudo generar exitosamente el código de la aplicación con los cambios solicitados, pero tuvo algunos inconvenientes al momento de crear el proyecto en el IDE, por lo que no pudo probar la aplicación. Posteriormente, durante la evaluación se pudo realizar la prueba faltante de la aplicación, obteniéndose excelentes resultados. El tiempo promedio de la actividad para este segundo grupo fue de 38.5 minutos. Este tiempo refleja los inconvenientes mencionados con uno de los participantes, pero vale la pena resaltar que el participante que pudo terminar exitosamente la actividad lo realizó en 20 minutos.

El promedio del puntaje ASQ obtenido en este grupo fue de 1.67. A diferencia del puntaje obtenido en el desarrollo manual, este puntaje refleja un nivel de satisfacción muy bueno por parte de los desarrolladores que realizaron las modificaciones a partir de nuestra propuesta.

Escenario	Tasa de Éxito Promedio	Tiempo de Finalización Promedio (minutos)	Puntaje ASQ Promedio
Modificaciones (MD2)	72%	38.5	1.67

Cuadro 6.7 Mediciones de usabilidad sobre el proceso de realizar una modificación a la aplicación generada: modificaciones al modelo

Todos de los participantes pudieron realizar las modificaciones a nivel código (cambiar el tipo de persistencia de la entidad), pero solamente 2/5 pudieron finalizar y probar los cambios realizados. A continuación resaltamos algunos puntos interesantes encontrados durante el análisis de ambos grupos de desarrollo:

- *Tiempo*: notamos una considerable diferencia comparando los tiempos de quienes pudieron culminar la actividad. Se puede decir que el desarrollo con nuestro enfoque permitió realizar la modificación 2.85X más rápido que en el desarrollo manual.
- *LoC de las modificaciones solicitadas*: comparando el desarrollo para la plataforma Android, encontramos una diferencia no despreciable: desarrollando manualmente se agregó un promedio de 309 líneas; con el enfoque MoWebA Mobile se agregaron 226 líneas.
- *ASQ*: el grupo que realizó manualmente la modificación de la aplicación, arrojó un puntaje promedio de 3 en ASQ; el grupo que trabajó con nuestro enfoque, arrojó una puntuación de 1.67. Ambos grupos experimentaron diferentes tipos

de problemas, por lo que estos puntajes reflejan el nivel de conformidad que tuvieron durante la actividad.

- Una misma modificación fue realizada por manualmente (modificaciones al código) y por el enfoque propuesto (modificaciones al modelo). Resaltamos que las modificaciones al modelo no necesitaron ninguna modificación manual extra, destacando la expresividad de nuestro modelo, en el contexto del ejercicio planteado.

Comentarios recolectados. El mayor problema destacado por los desarrolladores en esta actividad fue la falta de tiempo para poder terminar exitosamente los cambios solicitados, pero alentaron diciendo que con un poco más de práctica podrían realizarlo sin inconvenientes. Los desarrolladores que realizaron las modificaciones al código destacaron la legibilidad y la buena estructuración del código generado. Por otra parte, uno de los desarrolladores que realizó el proceso de modificación al modelo, nuevamente resaltó que tuvo inconvenientes con el proceso de importar las carpetas.

- O2*PI3: ¿Qué percepción de satisfacción presenta el enfoque MDD propuesto?*

  La percepción de satisfacción de MoWebA Mobile de parte de los desarrolladores tuvo en promedio un puntaje de 70 puntos, con desviación estándar del 15.2. Convirtiendo al rango percentil de Sauro, obtenemos un valor de 56%. Este resultado está por encima del promedio.

  En comparación con la percepción de satisfacción obtenida por los modeladores (50 puntos en el SUS), el puntaje de los desarrolladores es muy alta, pudiendo haber sido aún más elevada de no ser por los errores mencionados en O2*PI1* y O2*PI2*. Pero en general, analizando los cuestionarios SUS de los desarrolladores, y a diferencia de los modeladores, el enfoque propuesto fue muy bien aceptado, considerándolo consistente y poco complejo.

  Si bien, fue muy interesante y útil analizar la percepción de ambos perfiles por separado, creemos necesario llevar a cabo más pruebas, con ambos perfiles contemplando todo el proceso de desarrollo, y así obtener una percepción unificada de usabilidad general de nuestro enfoque.

Comentarios recolectados. Pudimos recabar la opinión de los desarrolladores respecto al desarrollo general con nuestro enfoque, y sobre la cobertura, de la aplicación generada, de los requerimientos iniciales establecidos. Muchos coincidieron en decir

que al no poder culminar exitosamente el proceso de modificación, les era difícil emitir una opinión sobre la validación de los requerimientos iniciales. En cuanto a las sugerencias y opiniones sobre MoWebA Mobile, destacaron que es un enfoque fácil de utilizar, que con algunos ajustes podrían lograrse cosas interesantes. Los ajustes mencionados incluían mejorar la importación del proyecto, inclusive, hacer el proceso automáticamente.

Errores encontrados en los cuestionarios SUS. En el SUS de la sesión 3 encontramos dos casos de datos corrompidos: un *valor perdido* y un *N/A*. Para solucionar recurrimos a una de las técnicas de la Minería de Datos para tratar valores o datos perdidos llamada *Mean Sustitution* [22], el cual establece completar el valor con el promedio de los valores encontrados. En nuestro caso, rellenamos los datos corrompidos con el promedio de la escala SUS = 3. Según Sauro [54], si un participante no puede responder a un ítem por alguna razón, deben seleccionar el punto central de la escala, lo que coincide con el valor arrojado por el *Mean Sustitution*.

- O2*PI4: ¿Qué percepción de portabilidad presenta el enfoque MDD propuesto?*

  Destacamos que las actividades del escenario sobre portabilidad (escenario P1), fueron realizadas exitosamente por todos los participantes. Los desarrolladores pudieron hacer pruebas de la aplicación y los mecanismos a disposición, mediante la carga de datos. Luego, hicieron las mismas pruebas en otros teléfonos móviles con diferentes sistemas operativos móviles.

  Mediante el cuestionario para obtener una primera aproximación a la portabilidad, pudimos obtener la percepción de los desarrolladores sobre la portabilidad de nuestro enfoque. A continuación hacemos un resumen de las respuestas recabadas:

  - *¿Notaste alguna diferencia con respecto a las funcionalidades?* En general, todos coincidieron en que no se presentaron problemas con la persistencia, pudieron hacer las pruebas satisfactoriamente con los distintos mecanismos de persistencia en las distintas plataformas. Sin embargo, resaltaron que sí hubo problemas con algunos proveedores de datos.
  - *¿Te parece útil MoWebA?* A todos les resultó útil la propuesta, resaltando que el ahorro de esfuerzo y tiempo en la generación de código, en este tipo de desarrollo, es notable.
  - *Los problemas en este escenario* fueron con los proveedores de datos. Estos inconvenientes se presentaron en algunos teléfonos, con sensores como el GPS

y el giróscopo, y con algunos *hardware* específicos como la cámara. Estas fallas probablemente se debieron a permisos faltantes en el teléfono. También reportaron algunos errores de interfaz, como algunos títulos equivocados. Por último, se registró una diferencia en la aplicación, en Windows Phone se generaba la pantalla de prueba para el audio, pero en Android no.

En general fue muy positiva esta actividad, confirmando a base de pruebas con los desarrolladores, que los mecanismos de persistencia funcionaban correctamente en las distintas plataformas. Resaltamos que sí hubo inconvenientes con algunos proveedores de datos, y algunos detalles de interfaz, que nos desafía a seguir mejorando y solucionando detalles, a fin de lograr un enfoque más robusto.

Resaltamos que el objetivo con estas pruebas realizadas fue obtener una primera aproximación sobre la validación de la portabilidad de nuestro enfoque, en pos de llevar a cabo validaciones más formales y/o experimentos (fuera del alcance de nuestro PFC).

6.2 Comparativa del desarrollo manual y automático de aplicaciones móviles

La primera etapa del desarrollo de nuestra propuesta se basó en el desarrollo manual del prototipo de la aplicación que generaríamos. Este prototipo sirvió como guía para identificar los distintos componentes a generar, y así diseñar las reglas de transformación. El desarrollo móvil fue realizado por un desarrollador con experiencia en Android. Sin embargo, el mismo ha realizado el desarrollo para las plataformas Android y Windows Phone, plataforma con la cual no tenía experiencia. En el cuadro 6.8 resumimos la lista de todas las actividades involucradas en el desarrollo de la aplicación por plataforma, y los tiempos involucrados. Se puede notar la diferencia de tiempo total de desarrollo, notablemente más alta para Windows Phone. Esto se debió a la curva de aprendizaje que fue requerida para trabajar con dicha plataforma.

A partir del tiempo de desarrollo manual recabado, en el cuadro 6.9 resumimos los tiempos de desarrollo por plataforma en los distintos enfoques. El tiempo total de desarrollo manual (para Android y Windows Phone) fue de 5160 minutos. Por otra parte, el tiempo total promedio de desarrollo con MoWebA Mobile, recabado a partir de la experiencia de validación realizada, fue de 281 minutos. Sin dudar, es notoria la diferencia en cuanto al tiempo de desarrollo. A parte de esto, si nos fijamos en el cuadro 6.2, en los escenarios M1,

Módulos	Lista de actividades	Android (horas)	Windows (horas)
Base de datos	Trabajo con SQLite en la plataforma (Aprendizaje de la herramienta y/o configuraciones iniciales)	0	5
	Diseño e implementación de pantallas	1	3
	Creación de la BD y Tablas	1	1.5
	Generación de los DAOs para cada tabla	2	3
	Generación de beans	0.5	1
	Testing	0.5	0.5
Files	Trabajo con Files en la plataforma (Aprendizaje de la herramienta y/o configuraciones iniciales)	0	3
	Diseño e implementación de pantallas	1	1
	Diseño de los Helpers	0.5	2
	Testing	0.25	0.5
Pares clave - valor	Trabajo con pares clave valor en la plataforma (Aprendizaje de la herramienta y/o configuraciones iniciales)	0	6
	Diseño e implementación de pantallas	1	1
	Diseño de los Helpers	0.5	2
	Testing	0.25	1
REST	Trabajo con la librería (Aprendizaje de la herramienta y/o configuraciones iniciales)	0	6.5
	Diseño de la interfaz de API y servicios	0.5	0.5
	Diseño e implementación de pantallas	0.5	1
	Testing (diseño de servicios de prueba)	2	2
Sensores	Trabajo con sensores (Aprendizaje de la herramienta y/o configuraciones iniciales)	5	7
	Diseño e implementación de pantallas	3	3.5
	Testing	0.5	2
Interoperabilidad con otras aplicaciones	Trabajo con el método (Aprendizaje de la herramienta y/o configuraciones iniciales)	4	5
	Diseño e implementación de pantallas	0	1
	Testing	1	2
	Total	25	61

Cuadro 6.8 Lista de actividades para el desarrollo del prototipo de la aplicación *e-market*

	MoWebA Mobile					Desarrollo manual		
	Capacitación	**Modelado**	**Generación de código**	**Generación de la aplicación**	**Tiempo Total**	**Aplicación para Android**	**Aplicación para Windows Phone**	**Tiempo Total**
Tiempo (minutos)	150	64.17	15.33	51.6	281.1	1500	3660	5160

Cuadro 6.9 Tiempos de desarrollo con el enfoque manual y el enfoque MoWebA Mobile

GC1 y GP1, podemos ver que el desarrollo con nuestro enfoque involucra menos pasos que el presentado en el cuadro 6.8.

Otros aspectos interesantes a destacar, mencionado en el análisis de la pregunta de investigación O2*PI2*, son las evidencias de una importante diferencia al momento de realizar modificaciones siguiendo ambos enfoques. Con MoWebA Mobile se pudo notar una ganancia de tiempo de realización de las modificaciones y una no despreciable diferencia en cuanto a la cantidad de líneas de código agregadas.

Son importantes los puntos a favor de MoWebA Mobile en comparación al desarrollo manual. El desarrollo manual involucra una curva de aprendizaje por plataforma, junto con el tiempo de desarrollo por cada plataforma destino. Con MoWebA Mobile encontramos una curva de aprendizaje en cuanto al modelado (utilización del perfil móvil), pero sin la necesidad de conocimiento previo por plataforma móvil. Si bien, la aplicación generada está enfocada solamente en la capa de datos, el código generado para ambas plataformas junto con las funcionalidades y utilidades para el manejo de la persistencia y los proveedores de datos identificados, resulta de mucha ayuda al iniciar proyectos de desarrollo móvil enfocados en este tipo de aplicaciones.

6.3 Síntesis del capítulo

El propósito de la experiencia de validación presentada en este capítulo fue realizar un primer análisis de la propuesta para obtener un primer juicio de intuiciones, en cuanto a su usabilidad y portabilidad, que nos permita encaminar a futuros trabajos de validaciones, como experimentos y/o casos de estudio. Si bien, seguimos las actividades de un caso de estudio, la experiencia de validación realizada no constituyó un caso de estudio ya que no fue aplicado dentro de un contexto real.

Evaluamos la usabilidad de nuestro enfoque desde el punto de vista de los modeladores y desarrolladores móviles, a fin de obtener primeras valoraciones sobre la experiencia del usuario final, que nos ayuden a realizar mejoras y correcciones, de manera a ir obteniendo un enfoque cada vez más estable, consistente, confiable y fácil de usar. De la misma forma, nos pareció interesante tener una primera aproximación de validación de la portabilidad de nuestro enfoque, de cara a preparar futuras validaciones más formales. Señalamos el problema de la fragmentación, el cual incrementa el esfuerzo de desarrollo de aplicaciones móviles, incluyendo la utilización de la persistencia en estos, debido a que las diferentes plataformas móviles manejan de manera distinta los mecanismos de persistencia. Para ello trabajamos en recopilar los juicios de los desarrolladores móviles respecto a este aspecto de calidad del software.

Los resultados de usabilidad de nuestro enfoque en general fueron buenos. Todas las etapas de desarrollo con MoWebA Mobile arrojaron buenos resultados de satisfacción de usabilidad. La generación de código y el proceso de modificación de la aplicación a partir de MoWebA Mobile, arrojaron muy buenos resultados. No sucedió así con el proceso de modificación manual de la aplicación, el cual arrojó el resultado más elevado, seguido del proceso de modelado.

Para hablar de la percepción de satisfacción general de nuestro enfoque, tenemos que hablar de dos puntos de vista: la de los modeladores y desarrolladores. Por una parte, los modeladores percibieron un poco de complejo el enfoque, sintiéndose algo inseguros en la utilización de la misma y con la necesidad de aprender muchas cosas para poder manejar el enfoque. Creemos que esto pudo deberse a la primera experiencia de trabajo que tuvieron con MoWebA Mobile, dándonos indicios de que para minimizar la curva de aprendizaje podrían requerirse más clases de preparación. Esto nos llama la atención puesto que se brindaron materiales informativos, e incluso se redujo la complejidad del modelado. Por otra parte, el enfoque propuesto fue muy bien aceptado por los desarrolladores, considerándolo consistente y poco complejo. Si bien, fue muy interesante y útil analizar la percepción de ambos perfiles por separado, creemos necesario llevar a cabo más pruebas, con ambos perfiles contemplando todo el proceso de desarrollo, y así obtener una percepción unificada de usabilidad general de nuestro enfoque.

Las pruebas hacia las primeras aproximaciones de validación de la portabilidad resultaron muy positivas. Las pruebas con los desarrolladores nos permitieron verificar que los mecanismos de persistencia funcionaban correctamente, expandiendo las pruebas a mayor número de versiones de sistemas operativos móviles. Resaltamos que sí hubo inconvenientes con algunos proveedores de datos, y algunos detalles de interfaz, que nos desafía a seguir mejorando y solucionando detalles, a fin de lograr un enfoque más robusto.

Por último, comparando el desarrollo manual de aplicaciones móviles con MoWebA Mobile, en general afirmamos que, en el contexto de las pruebas realizadas, el desarrollo con MoWebA Mobile involucra menos pasos. Además, con MoWebA Mobile encontramos una curva de aprendizaje en cuanto al modelado (para la utilización del perfil ASM móvil definido), pero sin la necesidad de conocimiento previo por plataforma móvil. Si bien, la aplicación generada está enfocada solamente en la capa de datos, el código generado para ambas plataformas resulta de mucha ayuda al iniciar proyectos de desarrollo móvil enfocados en este tipo de aplicaciones. De la misma forma, al realizar modificaciones tanto de forma manual y con MoWebA Mobile, resaltamos que MoWebA Mobile tuvo mejor percepción de usabilidad por parte de los desarrolladores. MoWebA Mobile permitió realizar los cambios mucho más rápido que haciéndolos manualmente, además de agregar menos líneas. Destacamos la expresividad de nuestro enfoque, puesto que las modificaciones al modelo no necesitaron ninguna modificación manual extra.

Capítulo 7

Conclusión

En este Proyecto Final de Carrera (PFC) nos hemos planteado como objetivo general definir un enfoque MDD, mediante la extensión de MoWebA, para el desarrollo de aplicaciones móviles nativas enfocadas en la capa de datos.

Los teléfonos móviles inteligentes cuentan con avanzados recursos computacionales, permitiendo correr aplicaciones bajo un avanzado sistema operativo. Entre los sistemas operativos más utilizados actualmente se encuentra Android. Estas aplicaciones móviles son dirigidas por eventos, interactúan con otras aplicaciones, manejan sensores y hardware específicos del teléfono y trabajan con recursos limitados como memoria, batería, conectividad, entre otros.

El ambiente fragmentado es uno de los principales retos para los desarrolladores de aplicaciones móviles en la actualidad, donde la variedad de plataformas origina este fenómeno conocido como fragmentación. El reto de la generación multiplataforma se encuentra en el desarrollo de aplicaciones nativas, donde desarrollar para cada plataforma implica esfuerzo, costo de mantenimiento y requiere de cierto nivel de experiencia. Existen herramientas que (semi)automatizan el desarrollo multiplataforma de aplicaciones móviles, hablamos de los *frameworks* y soluciones MDD. Una comparación entre estos, nos indicó que un enfoque dirigido por modelos podría constituirse en una opción factible y adecuada contra la fragmentación, al estar más orientada al desarrollo de aplicaciones nativas que los *frameworks*, específicamente en el dominio de los móviles. Con MDD, los desarrolladores de aplicaciones, describen sus aplicaciones en un nivel alto de abstracción mediante modelos independientes de detalles específicos de la plataforma, tanto para la presentación, la lógica de negocios y el acceso de datos. Esta especificación es traducida a código para diferentes plataformas. Siguiendo el enfoque MDA, significa implementar PIM para los diversos aspectos de los móviles (modelos independientes de aspectos tecnológicos), y luego mediante

transformaciones sucesivas, llegar al PSM para cada plataforma a la que se desea implementar la aplicación.

Otro reto presente en el desarrollo de aplicaciones móviles constituye manejar los recursos limitados de los teléfonos móviles. Mencionamos el problema de la naturaleza transitoria de los teléfonos móviles, donde la conectividad implica un cuidado especial, sobre todo en el desarrollo de aquellas aplicaciones móviles que están en constante conexión a Internet realizando peticiones a servicios web. Denominamos a estos tipos de aplicaciones como aplicaciones orientadas a datos. En estos, la persistencia de datos o el almacenamiento local, es una necesidad en caso que no se cuente con esta conexión.

En la actualidad existen diferentes opciones de almacenamiento, como HTML5 (para aplicaciones híbridas), el almacenamiento en servicios de alojamiento en la nube (como Google Drive y Amazon S3) y las bases de datos. Categorizando estas opciones encontramos que los mecanismos de almacenamiento local más comunes en los teléfonos móviles son los pares clave-valor, archivos, base de datos integradas y almacenamiento en dispositivos externos (por ejemplo, tarjetas de memoria SD). Todos estos mecanismos tienen soporte en los diferentes sistemas operativos móviles, pero cada una lo maneja de manera distinta y particular, notándose así que el problema de la fragmentación también provoca dificultades al momento de trabajar con el diseño de la persistencia para una aplicación móvil.

Un estudio de la utilización de la persistencia en el desarrollo de las aplicaciones móviles, desde el punto de vista de las propuestas MDD, nos permitió identificar la necesidad de especificaciones para describir la persistencia en el modelado de aplicaciones móviles y la poca adopción de MDA (en especial del estándar MOF). Es necesario cubrir conceptualmente la persistencia en el desarrollo de aplicaciones móviles, mediante elementos específicos que permitan tener en cuenta las distintas opciones y mecanismos de persistencia en el modelado, a fin de permitir el almacenamiento de datos incluso sin conexión de red, y considerando la conexión con fuentes remotas. Siguiendo la arquitectura multicapas, la capa de datos se encarga de cubrir los aspectos referentes a la persistencia en las aplicaciones móviles: persistencia y proveedores de datos.

MoWebA se presenta como una metodología MDD enfocada al desarrollo de aplicaciones web centrado en roles, bajo los estándares MDA. El ASM de MoWebA constituye uno de los aspectos más interesantes y aprovechables en busca de solucionar los problemas planteados: diseñar un ASM móvil que contemple todos los elementos conceptuales necesarios para cubrir el diseño de la capa de datos en las aplicaciones móviles. Teniendo en cuenta esto, presentamos nuestro enfoque MoWebA Mobile.

MoWebA Mobile se enfoca en el desarrollo de aplicaciones móviles nativas enfocadas en la capa de datos, abarcando los aspectos de persistencia y proveedores de datos. El proceso

de desarrollo consiste en utilizar el perfil móvil definido para generar modelos en MagicDraw, exportar estos en formato XMI a Acceleo, y generar código a partir de ellos utilizando las reglas de transformación definidas. El código generado debe ser compilado en los IDEs respectivos del sistema operativo móvil destino, para así obtener la aplicación final. Para la definición de este ASM móvil, partimos de MoWebA. En primer lugar, extendimos el PIM de entidades de MoWebA a fin de aprovechar elementos conceptuales del tipo estructural. Se extendió la propiedad de una entidad, introduciendo elementos tales como *datatype*, *id*, *size*, entre otros, los cuales enriquecen la definición estructural de este diagrama. En segundo lugar, definimos el metamodelo y perfil UML móvil a partir de las extensiones realizadas. Para la definición de la persistencia, se agregaron elementos como *persistentEntity*, *persistentType* y *persistentEntityProperty*, para poder definir una entidad persistente, establecer un tipo de mecanismo de persistencia para esa entidad y agregar propiedades a los atributos definidos. Para la definición de los proveedores de datos, las interfaces *WebServiceInterface*, *HardwareDeviceInterface* y *MobileAppDataInterface*, permiten definir los proveedores externos, internos y la interoperabilidad con otras aplicaciones. Por último, para este PFC, definimos las reglas de transformación para la obtención automática de los elementos definidos en el modelo. Estas reglas permiten generar aplicaciones completas que evidencian las funcionalidades definidas. Se genera tanto para Android como para Windows Phone.

A fin de evaluar MoWebA Mobile, llevamos a cabo la validación preliminar con la intención de realizar un primer análisis de la propuesta y así obtener un primer juicio de intuiciones en cuanto a su usabilidad y portabilidad. Evaluamos la usabilidad de nuestro enfoque desde el punto de vista de los modeladores y desarrolladores móviles, a fin de obtener primeras valoraciones sobre la experiencia del usuario final, que nos ayuden a realizar mejoras y correcciones, de manera a ir obteniendo un enfoque cada vez más estable, consistente, confiable y fácil de usar. De la misma forma, ante el problema mencionado de que la fragmentación dificulta el diseño de la persistencia ante tantos mecanismos de persistencia que son manejados de manera diferente en cada sistema operativo móvil, nos pareció interesante tener una primera aproximación de validación de la portabilidad de nuestro enfoque, de cara a preparar futuras validaciones más formales.

Los resultados de usabilidad de nuestro enfoque en general fueron buenos. Todas las etapas de desarrollo con MoWebA Mobile arrojaron buenos resultados de satisfacción de usabilidad. De la misma forma, las pruebas hacia las primeras aproximaciones de validación de la portabilidad resultaron muy positivos. Las pruebas con los desarrolladores nos permitieron verificar que los mecanismos de persistencia funcionaban correctamente, expandiendo las pruebas a mayor número de versiones de sistemas operativos móviles.

Un aspecto importante a destacar tras esta validación preliminar es que, la percepción de satisfacción general de nuestro enfoque tuvo dos puntos de vista: la del modelador y desarrollador. Por una parte, los modeladores percibieron un poco de complejo el enfoque, sintiéndose algo inseguro en la utilización de la misma y con la necesidad de aprender muchas cosas para poder manejar el enfoque. Por otra parte, el enfoque propuesto fue muy bien aceptado por los desarrolladores, considerándolo consistente y poco complejo. Si bien, fue muy interesante y útil analizar la percepción de ambos perfiles por separado, creemos necesario llevar a cabo más pruebas, con ambos perfiles contemplando todo el proceso de desarrollo, y así obtener una percepción unificada de usabilidad general de nuestro enfoque.

Destacamos que tras comparaciones del desarrollo manual de aplicaciones móviles con MoWebA Mobile, en general afirmamos que, en el contexto de las pruebas realizadas, el desarrollo con MoWebA Mobile involucra menos pasos. Además, con MoWebA Mobile encontramos una curva de aprendizaje en cuanto al modelado (utilización del perfil móvil), pero sin la necesidad de conocimiento previo por plataforma móvil. Si bien, la aplicación generada está enfocada solamente en la capa de datos, el código generado para ambas plataformas resulta de mucha ayuda al iniciar proyectos de desarrollo móvil enfocados en este tipo de aplicaciones. De la misma forma, al realizar modificaciones tanto de forma manual y con MoWebA Mobile, resaltamos que MoWebA Mobile tuvo mejor percepción de usabilidad por parte de los desarrolladores. MoWebA Mobile permitió realizar los cambios mucho más rápido que haciéndolos manualmente, además de agregar menos líneas. Destacamos la expresividad de nuestro enfoque, puesto que las modificaciones al modelo (con MoWebA Mobile) no necesitaron ninguna modificación manual extra.

En base a todo lo mencionado, decimos que MoWebA Mobile cumple con el objetivo establecido: diseñar aplicaciones móviles considerando la persistencia de datos. Esta propuesta, a parte de plantearse el almacenamiento local de datos mediante diferentes mecanismos, permite considerar los distintos proveedores de datos para una aplicación móvil. Además, considerando la arquitectura multicapas, conceptualmente se enmarca dentro de una capa de datos, permitiendo la expansión del enfoque hacia otras capas. En cuanto a la aplicación generada, podemos decir que mediante las reglas de transformación definidas, se pueden generar aplicaciones funcionales para Android y Windows Phone, con capacidad de trabajar en modo “sin conexión”, considerando los elementos definidos. La aplicación generada también considera aspectos de interfaz gráfica básica y provee funcionalidades útiles que facilitan la tarea del desarrollador móvil (por ejemplo, el esqueleto de las interfaces REST y sus conexiones, operaciones CRUD sobre la base de datos y clases de ayuda con funcionalidades que permiten manejar los distintos mecanismos de persistencia).

MoWebA Mobile tiene potencial de expandirse, de generar una aplicación más completa y que pueda solucionar más problemas relacionados a otras capas. Las pruebas realizadas fueron exitosas puesto que pudimos identificar puntos débiles a corregir, de cara a la obtención de una versión cada vez más estable y consistente de nuestro enfoque, y considerando los comentarios de los participantes de la experiencia, de manera a ir mejorando cada vez más.

7.1 Principales contribuciones

A partir de lo concluido previamente, podemos afirmar que los objetivos propuestos fueron alcanzados. A continuación mencionamos los principales aportes de nuestro PFC:

- Un estudio detallado acerca de las aplicaciones móviles, de la persistencia en los móviles, MDD y del enfoque MoWebA.
- Un estudio de las propuestas MDD para el desarrollo de aplicaciones móviles, observando los aspectos de persistencia utilizados.
- Una extensión de MoWebA para el desarrollo de aplicaciones móviles, considerando la capa de datos mediante metamodelos y perfiles ASM; y el establecimiento de reglas de transformación para la generación de una aplicación móvil funcional tanto para Android como Windows Phone, que permita evidenciar las funcionalidades definidas en el enfoque propuesto.
- Una validación de la propuesta a través de una experiencia de desarrollo de una aplicación móvil.

7.2 Trabajos futuros

Presentamos los trabajos futuros y directrices que pueden ser seguidas en futuras investigaciones relacionadas a este proyecto:

- **Generación automática de la aplicación**: se podría generar la aplicación final sin necesidad de compilarlo primero en un IDE. Implica generar todos los ejecutables y archivos temporales, atendiendo las características específicas de cada plataforma móvil.
- **Diseñar y evaluar aplicaciones de la industria**: las pruebas llevadas con MoWebA Mobile en este PFC, contemplaron el diseño de una aplicación móvil creada con los

propósitos de la experiencia. Podría pensarse realizar las pruebas de diseño de una aplicación ya existente en el mercado.

- **Generación para otras plataformas y arquitecturas**: actualmente MoWebA Mobile soporta la generación para Android y Windows Phone, por lo que sería interesante incluir la generación para iOS. Inclusive, se podría tener soporte para generar aplicaciones móviles web y aplicaciones móviles híbridas, las cuales se basan en tecnologías web (por ejemplo, HTML5, CSS y JavaScript). Por otra parte, podría aprovecharse el concepto de capa de datos definido con MoWebA Mobile, y expandirla a otras arquitecturas (por ejemplo, RIAs).

- **Expandir la generación de los servicios web**: actualmente solo generamos la interfaz de los servicios web para la arquitectura REST. Se podría pensar en contemplar el modelado completo de los servicios. Inclusive, podrían expandirse a otras arquitecturas (por ejemplo, SOAP).

- **Realizar validaciones más estrictas de la propuesta**: podrían llevarse a cabo experimentos formales, casos de estudio en un contexto industrial o comercial, y/o pruebas con mayor número de participantes. Estas pruebas formales deberían realizar el desarrollo completo con MoWebA Mobile, sin dividir en etapas de modelado y posterior generación de la aplicación, a fin de obtener una percepción general del enfoque.

- **Comparar las medidas de usabilidad**: se podría estudiar las medidas de usabilidad obtenidas utilizando el enfoque con las de otro enfoque o proceso desarrollando la misma aplicación.

- **Realizar validaciones considerando más aspectos de calidad del software**: en este PFC realizamos unas primeras aproximaciones con respecto a la portabilidad de nuestro enfoque. Podrían realizarse pruebas formales sobre este aspecto de calidad, e inclusive, podrían considerarse evaluar más aspectos de calidad del enfoque (por ejemplo, mantenibilidad y eficiencia).

Referencias

[1] Acerbis, R., Bongio, A., Brambilla, M., and Butti, S. (2015). *Engineering the Web in the Big Data Era: 15th International Conference, ICWE 2015, Rotterdam, The Netherlands, June 23-26, 2015, Proceedings*, chapter Model-Driven Development Based on OMG's IFML with WebRatio Web and Mobile Platform, pages 605–608. Springer International Publishing, Cham.

[2] Alamri, H. S. and Mustafa, B. A. (2014). Software engineering challenges in multi platform mobile application development. *Advanced Science Letters*, 20(10-12):2115–2118.

[3] Albert, W. and Tullis, T. (2013). *Measuring the user experience: collecting, analyzing, and presenting usability metrics*. Newnes.

[4] Balagtas-Fernandez, F., Tafelmayer, M., and Hussmann, H. (2010). Mobia modeler: easing the creation process of mobile applications for non-technical users. In *Proceedings of the 15th international conference on Intelligent user interfaces*, pages 269–272. ACM.

[5] Balagtas-Fernandez, F. T. and Hussmann, H. (2008). Model-driven development of mobile applications. In *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, ASE '08, pages 509–512, Washington, DC, USA. IEEE Computer Society.

[6] Bangor, A., Kortum, P., and Miller, J. (2009). Determining what individual sus scores mean: Adding an adjective rating scale. *Journal of usability studies*, 4(3):114–123.

[7] Barnett, S., Vasa, R., and Grundy, J. (2015a). Bootstrapping mobile app development. In *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, ICSE '15, pages 657–660, Piscataway, NJ, USA. IEEE Press.

[8] Barnett, S., Vasa, R., and Tang, A. (2015b). A conceptual model for architecting mobile applications. In *Software Architecture (WICSA), 2015 12th Working IEEE/IFIP Conference on*, pages 105–114.

[9] Basili, V. R. and Rombach, H. D. (1988). The tame project: Towards improvement-oriented software environments. *IEEE Transactions on software engineering*, 14(6):758–773.

[10] Botturi, G., Ebeid, E., Fummi, F., and Quaglia, D. (2013). Model-driven design for the development of multi-platform smartphone applications. In *Specification Design Languages (FDL), 2013 Forum on*, pages 1–8.

[11] Brambilla, M., Cabot, J., and Wimmer, M. (2012). *Model-Driven Software Engineering in Practice*. Morgan & Claypool Publishers.

[12] Brambilla, M., Mauri, A., and Umuhoza, E. (2014). *Mobile Web Information Systems: 11th International Conference, MobiWIS 2014, Barcelona, Spain, August 27-29, 2014. Proceedings*, chapter Extending the Interaction Flow Modeling Language (IFML) for Model Driven Development of Mobile Applications Front End, pages 176–191. Springer International Publishing, Cham.

[13] Brooke, J. et al. (1996). Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7.

[14] Charaf, H., Ekler, P., Mészáros, T., Kelényi, I., Kovari, B., Albert, I., Forstner, B., and Lengyel, L. (2014). Mobile platforms and multi-mobile platform development. *Acta Cybernetica*, 21:529–552.

[15] eMarketer (2015). Global media intelligence report executive summary. https://www.emarketer.com/public_media/docs/GMI-2015-ExecutiveSummary.pdf.

[16] Ferreira, J. A. L. and da Silva, A. R. (2014). Mobile cloud computing. *Open Journal of Mobile Computing and Cloud Computing*, 1(2).

[17] Fotache, M., Cogean, D., et al. (2013). Nosql and sql databases for mobile applications. case study: Mongodb versus postgresql. *Informatica Economica*, 17(2):41–58.

[18] Geiger-Prat, S., MarThín, B., España, S., and Giachetti, G. (2015). A gui modeling language for mobile applications. In *2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS)*, pages 76–87.

[19] Genero, M., Cruz-Lemus, J., and Piattini, M. (2014). Métodos de investigación en ingeniería del software. *RaMa. Shull, Forrest*.

[20] González, M., Cernuzzi, L., and Pastor, O. (2016). A Navigational Role-Centric Model Oriented Web Aproach - MoWebA. *International Journal of Web Engineering and Technology (IJWET)*, 11(1):29–67. Online.

[21] Grønli, T.-M., Hansen, J., Ghinea, G., and Younas, M. (2014). Mobile application platform heterogeneity: Android vs windows phone vs ios vs firefox os. In *Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on*, pages 635–641. IEEE.

[22] Han, J. and Kamber, M. (2006). *Data Mining: Concepts and Techniques*. The Morgan Kaufmann series in data management systems. Elsevier.

[23] Heitkötter, H. and Majchrzak, T. A. (2013). *Design Science at the Intersection of Physical and Virtual Design: 8th International Conference, DESRIST 2013, Helsinki, Finland, June 11-12, 2013. Proceedings*, chapter Cross-Platform Development of Business Apps with MD2, pages 405–411. Springer Berlin Heidelberg, Berlin, Heidelberg.

[24] Heitkötter, H., Majchrzak, T. A., and Kuchen, H. (2013). Cross-platform model-driven development of mobile applications with md 2. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 526–533. ACM.

[25] Hemel, Z. and Visser, E. (2011). *Declaratively programming the mobile web with Mobl*, volume 46. ACM.

[26] Holzinger, A., Treitler, P., and Slany, W. (2012). Making apps useable on multiple different mobile platforms: On interoperability for business application development on smartphones. In *Multidisciplinary research and practice for information systems*, pages 176–189. Springer.

[27] IbikunleF.A and A.A, A. (2013). Management issues and challenges in mobile database system. *International Journal of Engineering Sciences & Emerging Technologies*, 5:6.

[28] Iso, I. (2011). Iec25010: 2011 systems and software engineering–systems and software quality requirements and evaluation (square)–system and software quality models. *International Organization for Standardization*, 34:2910.

[29] Iso, W. (1998). 9241-11. ergonomic requirements for office work with visual display terminals (vdts). *The international organization for standardization*, 45.

[30] Jones, C. and Jia, X. (2014). The axiom model framework: Transforming requirements to native code for cross-platform mobile applications. In *Evaluation of Novel Approaches to Software Engineering (ENASE), 2014 International Conference on*, pages 1–12.

[31] Keele, S. (2007). Guidelines for performing systematic literature reviews in software engineering. In *Technical report, Ver. 2.3 EBSE Technical Report. EBSE*.

[32] Ko, M., Seo, Y. J., Min, B. K., Kuk, S., and Kim, H. S. (2012). Extending uml meta-model for android application. In *Computer and Information Science (ICIS), 2012 IEEE/ACIS 11th International Conference on*, pages 669–674.

[33] Kramer, D., Clark, T., and Oussena, S. (2010). Mobdsl: A domain specific language for multiple mobile platform deployment. In *Networked Embedded Systems for Enterprise Applications (NESEA), 2010 IEEE International Conference on*, pages 1–7.

[34] Lachgar, M. and Abdali, A. (2014). Generating android graphical user interfaces using an mda approach. In *2014 Third IEEE International Colloquium in Information Science and Technology (CIST)*, pages 80–85.

[35] LACHGAR, M. and ABDALI, A. (2015). Modeling and generating the user interface of mobile devices and web development with dsl. *Journal of Theoretical & Applied Information Technology*, 72(1).

[36] Le Goaer, O. and Waltham, S. (2013). Yet another dsl for cross-platforms mobile development. In *Proceedings of the First Workshop on the Globalization of Domain Specific Languages*, GlobalDSL '13, pages 28–33, New York, NY, USA. ACM.

[37] Lewis, J. R. (1991). Psychometric evaluation of an after-scenario questionnaire for computer usability studies: the asq. *ACM SIGCHI Bulletin*, 23(1):78–81.

[38] Mahmoud, Q. H., Zanin, S., and Ngo, T. (2012). Integrating mobile storage into database systems courses. In *Proceedings of the 13th Annual Conference on Information Technology Education*, SIGITE '12, pages 165–170, New York, NY, USA. ACM.

[39] Majchrzak, T. A., Ernsting, J., and Kuchen, H. (2015). Model-driven cross-platform apps: Towards business practicability. In *Proc. of the 27th Conference on Advanced Information Systems Engineering (CAiSE) Forum. CEUR*.

[40] Marinho, E. H. and Resende, R. F. (2015). Native and multiple targeted mobile applications. In *Computational Science and Its Applications–ICCSA 2015*, pages 544–558. Springer.

[41] Melman, C. (2014). A generative approach for data synchronization between web and mobile applications. Master thesis, Electrical Engineering, Mathematics and Computer Science.

[42] Min, B.-K., Ko, M., Seo, Y., Kuk, S., and Kim, H. S. (2011). A uml metamodel for smart device application modeling based on windows phone 7 platform. In *TENCON 2011 - 2011 IEEE Region 10 Conference*, pages 201–205.

[43] Muñoz Riesle, R., Marín, B., and López Cuesta, L. (2015). Applying iso 9126 metrics to mdd projects. In *ICSEA 2015: the Tenth International Conference on Software Engineering Advances, November 15-20, 2015, Barcelona, Spain*, pages 326–332.

[44] Nori, A. K. (2007). Mobile and embedded databases, sigmod 2007. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data, Beijing, China, disponible en: http://citeseerx. ist. psu. edu/viewdoc/summary*.

[45] Palmieri, M., Singh, I., and Cicchetti, A. (2012). Comparison of cross-platform mobile development tools. In *Intelligence in Next Generation Networks (ICIN), 2012 16th International Conference on*, pages 179–186. IEEE.

[46] Patterns, M. (2009). *Microsoft Application Architecture Guide*. Microsoft Press, 2nd edition.

[47] Pons, C., Giandini, R., and Perez, G. (2010). *Desarrollo de software dirigido por modelos : conceptos teóricos y su aplicación practica*. Mc Graw Hill Educacion, 1era edition.

[48] Redda, Y. A. (2012). Cross platform mobile applications development. *Norwegian University of Science and Technology, Norwegian University of Science and Technology, Norwegian University of Science and Technology, Master in Information Systems*.

[49] Ribeiro, A. and da Silva, A. R. (2012). Survey on cross-platforms and languages for mobile apps. In *Quality of Information and Communications Technology (QUATIC), 2012 Eighth International Conference on the*, pages 255–260. IEEE.

[50] Ribeiro, A. and da Silva, A. R. (2014a). Development of mobile applications using a model-driven software development approach.

[51] Ribeiro, A. and da Silva, A. R. (2014b). Xis-mobile: A dsl for mobile applications. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, SAC '14, pages 1316–1323, New York, NY, USA. ACM.

[52] Sabraoui, A., El Koutbi, M., and Khriss, I. (2013). A mda-based model-driven approach to generate gui for mobile applications. *International Review on Computers and Software (IRECOS)*, 8(3):845–852.

[53] Sanchiz, E., González, M., Aquino, N., and Cernuzzi, L. (2016). Development of mobile cloud applications through the model driven approach: A systematic mapping study. *CLEI Electronic Journal (CLEIej)*, 14(1). (to be published).

[54] Sauro, J. and Lewis, J. R. (2016). *Quantifying the user experience: Practical statistics for user research*. Morgan Kaufmann.

[55] Sommer, A. and Krusche, S. (2013). Evaluation of cross-platform frameworks for mobile applications. In *Software Engineering (Workshops)*, pages 363–376.

[56] Stahl, T., Voelter, M., and Czarnecki, K. (2006). *Model-driven software development: technology, engineering, management*. John Wiley & Sons.

[57] Usman, M., Iqbal, M., and Khan, M. (2014). A model-driven approach to generate mobile applications for multiple platforms. In *Software Engineering Conference (APSEC), 2014 21st Asia-Pacific*, volume 1, pages 111–118.

[58] Vaupel, S., Taentzer, G., Harries, J. P., Stroh, R., Gerlach, R., and Guckert, M. (2014). *Model-Driven Engineering Languages and Systems: 17th International Conference, MODELS 2014, Valencia, Spain, September 28 – October 3, 2014. Proceedings*, chapter Model-Driven Development of Mobile Applications Allowing Role-Driven Variants, pages 1–17. Springer International Publishing, Cham.

[59] Vique, R. R. (2012). Métodos para el desarrollo de aplicaciones móviles. *PID_00176755*.

[60] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2012). *Experimentation in software engineering*. Springer Science & Business Media.

[61] Xanthopoulos, S. and Xinogalos, S. (2013). A comparative analysis of cross-platform development approaches for mobile applications. In *Proceedings of the 6th Balkan Conference in Informatics*, BCI '13, pages 213–220, New York, NY, USA. ACM.

[62] Yin, R. K. (2013). *Case study research: Design and methods*. Sage publications.

[63] Yu, W., Amjad, T., Goel, H., and Talawat, T. (2008). An approach of mobile database design methodology for mobile software solutions. In *Grid and Pervasive Computing Workshops, 2008. GPC Workshops '08. The 3rd International Conference on*, pages 138–144.

Anexo A

Desarrollo dirigido por modelos de aplicaciones móviles

Propuestas seleccionadas como resultado del estudio de la literatura de soluciones MDD para el desarrollo de aplicaciones móviles (sección 4.2.1)

Nro	Título del documento	Año de publicación	Tipo de documento	Descripción
P1	A Model driven Approach to Generate Mobile Applications for Multiple Platforms [57]	2014	INPROCEEDINGS	Perfil UML para desarrollo de móviles multiplataforma. El enfoque se centra y permite generar código de lógica de negocios para varias plataformas. Desarrollo de prototipo de herramienta para la generación de código llamada MAG (Generador de Aplicaciones Móviles).
P2	Modelling and generating the user interface of mobile devices and web development with DSL [35]	2015	ARTICLE	Enfoque para el desarrollo de interfaz de usuario para aplicaciones móviles, aplicado a Android y a Java Server Faces Framework. Se define un lenguaje para desarrollo de interfaces gráficas, Technology Neutral DSL, con la intención de generar código nativo para varias plataformas.
P3	Generating Android graphical user interfaces using an MDA approach [34]	2014	INPROCEEDINGS	Enfoque para el desarrollo de interfaz de usuario para aplicaciones móviles, aplicado a Android. Se define un lenguaje para desarrollo de interfaces gráficas, Technology Neutral DSL, con la intención de generar código nativo para varias plataformas.
P4	Model Driven Development of Mobile Applications Allowing Role Driven Variants [58]	2014	INBOOK	Lenguaje de modelado para el desarrollo MDD de aplicaciones móviles según roles de usuario, basado en EMF. Prototipo de infraestructura MDD para implementación del lenguaje presentado, basado en GMF (editor gráfico) y generador de código.
P5	Extending the Interaction Flow Modeling Language IFML for Model Driven Development of Mobile Aplications Front End [12]	2014	INBOOK	Enfoque dirigido por modelos para el desarrollo de aplicaciones móviles, basadas en el estándar IFML. Se propone una extensión de IFML para móviles. Se presenta el desarrollo del prototipo de editor de modelos en Eclipse y el generador de código multiplataforma.
P6	Model-Driven Development Based on OMG's IFML with WebRatio Web and Mobile Platform [1]	2015	INBOOK	Propuesta de plataforma WebRatio para el desarrollo MDD de aplicaciones web y móviles basado en IFML.
P7	MobDSL A Domain Specific Language for multiple mobile platform deployment [33]	2010	INPROCEEDINGS	MobDSL para el desarrollo multiplataforma de aplicaciones móviles, basado en el lenguaje calculus.
P8	XIS-Mobile A DSL for Mobile Applications [51]	2014	INPROCEEDINGS	Especificación de aplicaciones móviles independientes de la plataforma, mediante lenguaje XIS-Mobile y framework, basados en perfiles UML

Cuadro A.1 Propuestas recopiladas para el desarrollo de aplicaciones móviles siguiendo el enfoque MDD (Parte I)

Nro	Título del documento	Año de publicación	Tipo de documento	Descripción
P9	Bootstrapping Mobile App Development [7]	2015	INPROCEEDINGS	Desarrollo de RAPPT, una herramienta que genera el esqueleto de aplicación móvil para Android, especificado mediante un DSL. Su objetivo es brindar mayor soporte para los desarrolladores y reducir la redundancia de tareas que se presentan utilizando los IDE actuales.
P10	Model-Driven Design for the Development of Multi-Platform Smartphone Applications [10]	2013	ARTICLE	Metodología de Diseño Dirigido por Modelos para el desarrollo de aplicaciones móviles independientes de la plataforma específica. Utiliza perfil UML2 para el PIM.
P11	A GUI Modeling Language for Mobile Applications [18]	2015	INPROCEEDINGS	Método para modelar interfaces móviles, como parte de un futuro proyecto de desarrollo MDD para aplicaciones móviles. Se presenta el lenguaje desarrollado: MIM (Modelado de Interfaz Móvil), y se evalúa la factibilidad de aplicación de la propuesta.
P12	The AXIOM Model Framework Transforming Requirements to Native Code for Cross-platform Mobile Applications [30]	2014	INPROCEEDINGS	Enfoque dirigido por modelos para el desarrollo multiplataforma de aplicaciones móviles que usa Axiom DSL para definir aplicaciones independientes de la plataforma. Desarrollo de prototipo Axiom para la generación de aplicaciones.
P13	Yet Another DSL for Cross-Platforms Mobile Development [36]	2013	INPROCEEDINGS	Presentación de XMOB DSL (Technology Neutral DSL), enfocado en aplicaciones móviles y basado en MDA.
P14	Model-Driven Cross-Platform Apps Towards Business Practicability [39]	2015	INPROCEEDINGS	MD2 como una solución para cumplir requisitos típicos de aplicaciones empresariales.
P15	Cross-Platform Development of Business Apps with MD2 [24]	2013	INBOOK	MD2 es un *framework* para el desarrollo MDD multiplataforma. Consiste en un DSL para describir aplicaciones de negocio y a partir de ahí, mediante generadores, crear automáticamente aplicaciones iOS y Android.
P16	A MDA-Based Model-Driven Approach to Generate GUI for Mobile Applications [52]	2013	ARTICLE	Enfoque MDA para modelado de GUI móvil. Presenta una forma de diseñar utilizando UML.
P17	A UML Metamodel for Smart Device Application Modeling based on Windows Phone 7 Platform [42]	2011	INPROCEEDINGS	Metamodelo UML extendido para el desarrollo de aplicaciones para Windows Phone 7.
P18	Extending UML Metamodel for Android Application [32]	2012	INPROCEEDINGS	Metamodelo UML extendido para el desarrollo de aplicaciones para Android

Cuadro A.2 Propuestas recopiladas para el desarrollo de aplicaciones móviles siguiendo el enfoque MDD (Parte II)

Anexo B

Aplicación *e-market*

Como definimos en la sección 5.3, la aplicación *e-market* consiste en una tienda virtual que hace entregas a domicilio, teniendo como requerimientos utilizar el GPS del teléfono, y además debe funcionar en modo "sin conexión". A partir de los modelos definidos con el ASM móvil de MoWebA Mobile, en la sección 5.4 explicamos qué código se genera a partir de las reglas de transformación definidas. En la siguiente sección se presenta la aplicación generada, mostrando las pantallas principales generadas, y luego, vemos parte del código generado teniendo en cuenta los distintos elementos modelados.

B.1 Pantallas de la aplicación

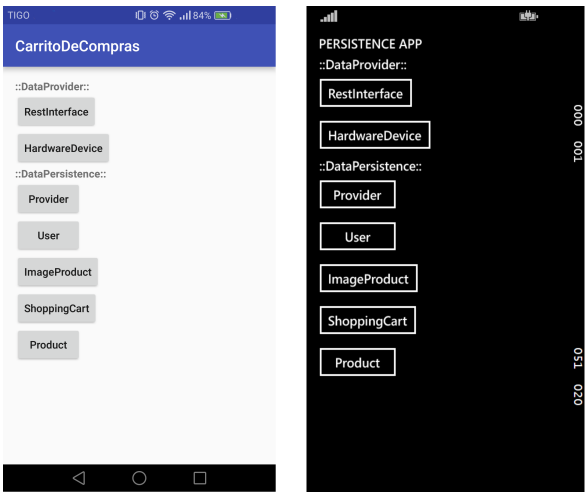


Figura B.1 Menú principal. Aplicación *e-market* para compras *online*. *A la izquierda Android, a la derecha Windows Phone*

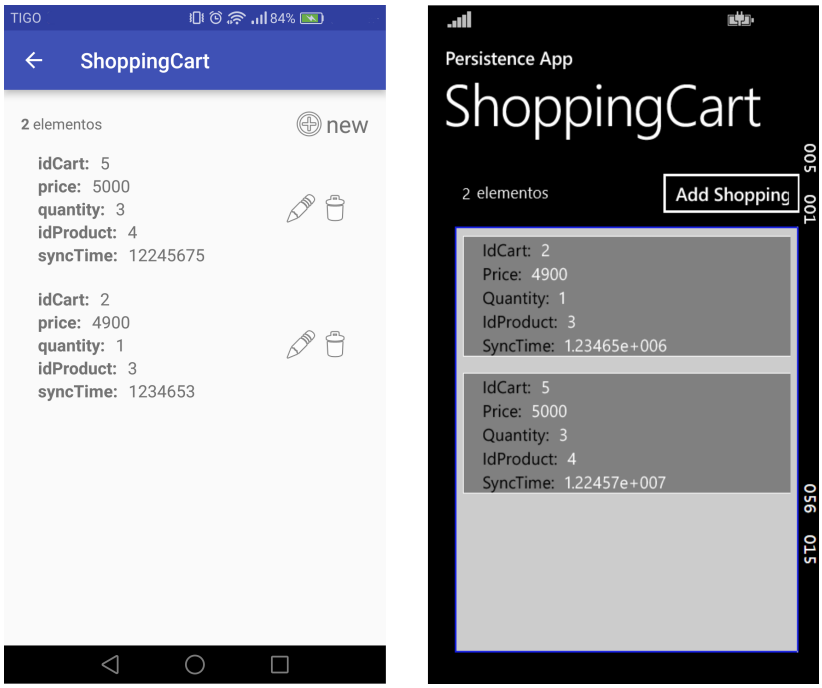


Figura B.2 Lista de datos de la tabla *ShoppingCart*. Aplicación *e-market*. *A la izquierda Android, a la derecha Windows Phone*

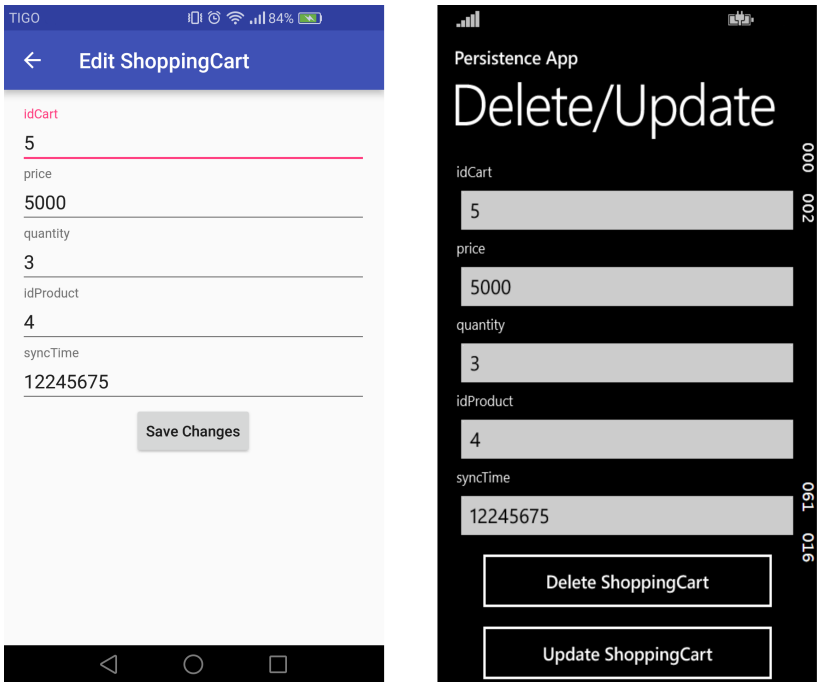


Figura B.3 Edición de los datos de la tabla *ShoppingCart*. Aplicación *e-market*. *A la izquierda Android, a la derecha Windows Phone*

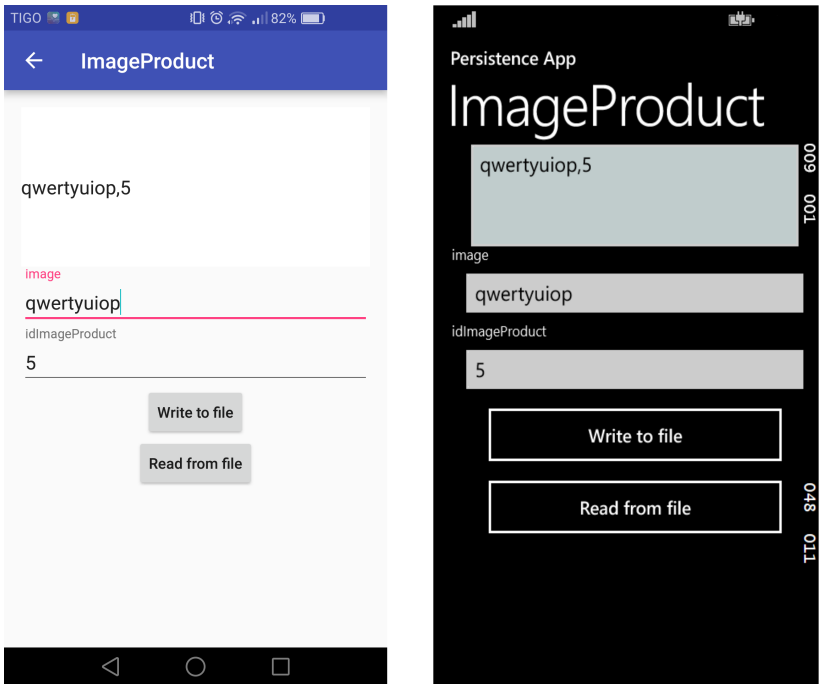


Figura B.4 Manejo de archivos para la entidad persistente *ImageProduct*. Aplicación *e-market*. *A la izquierda Android, a la derecha Windows Phone*

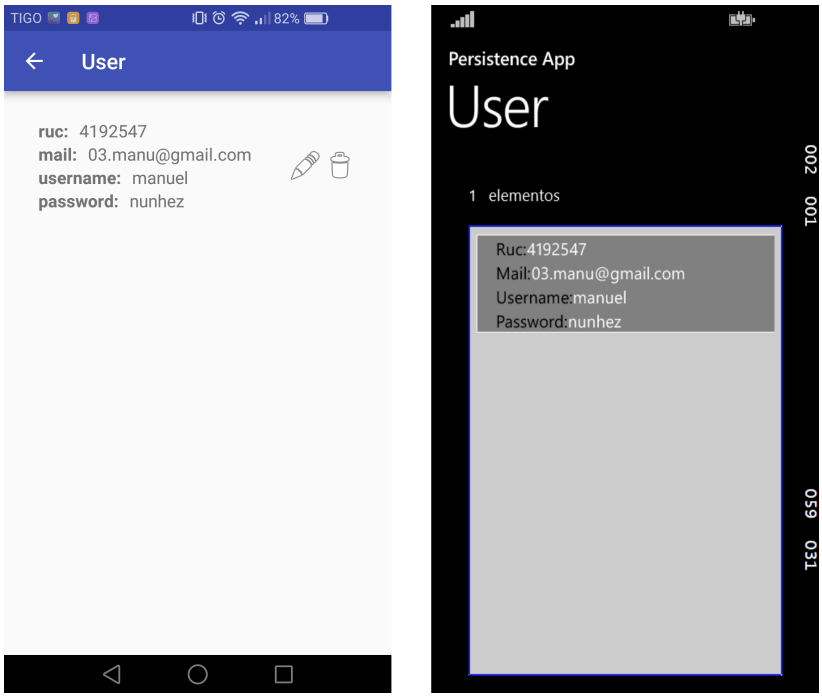


Figura B.5 Lista de datos almacenados en forma de pares clave-valor, para la entidad persistente *User*. Aplicación *e-market*. *A la izquierda Android, a la derecha Windows Phone*

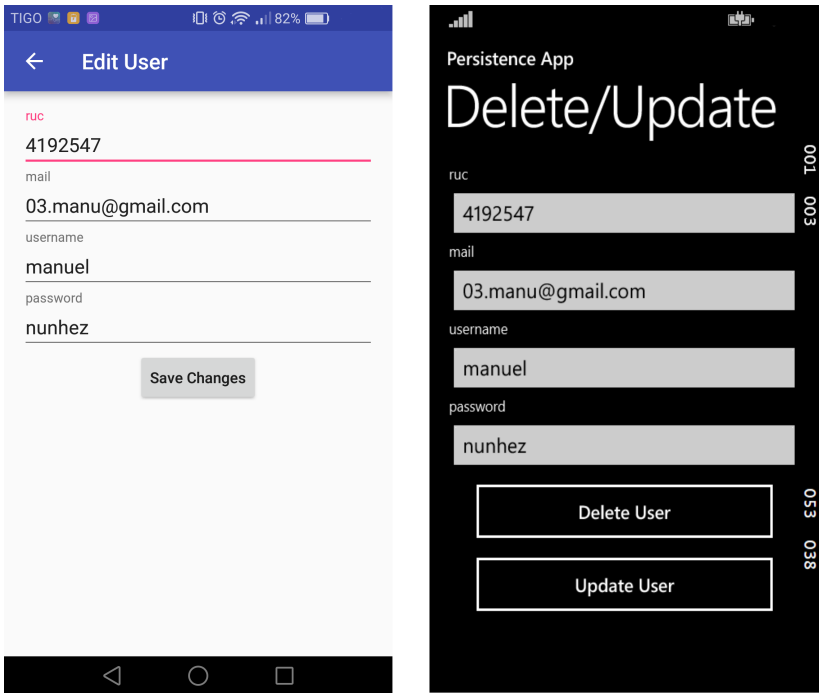


Figura B.6 Edición de los datos almacenados mediante pares clave-valor, para la entidad persistente *User*. Aplicación *e-market*. *A la izquierda Android, a la derecha Windows Phone*

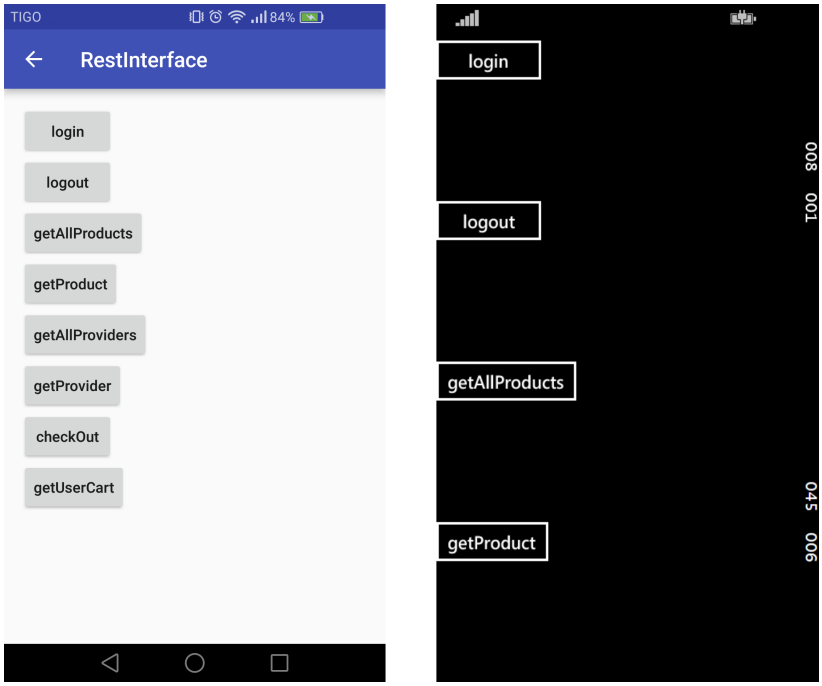


Figura B.7 *Endpoints* definidos. Aplicación *e-market*. *A la izquierda Android, a la derecha Windows Phone*

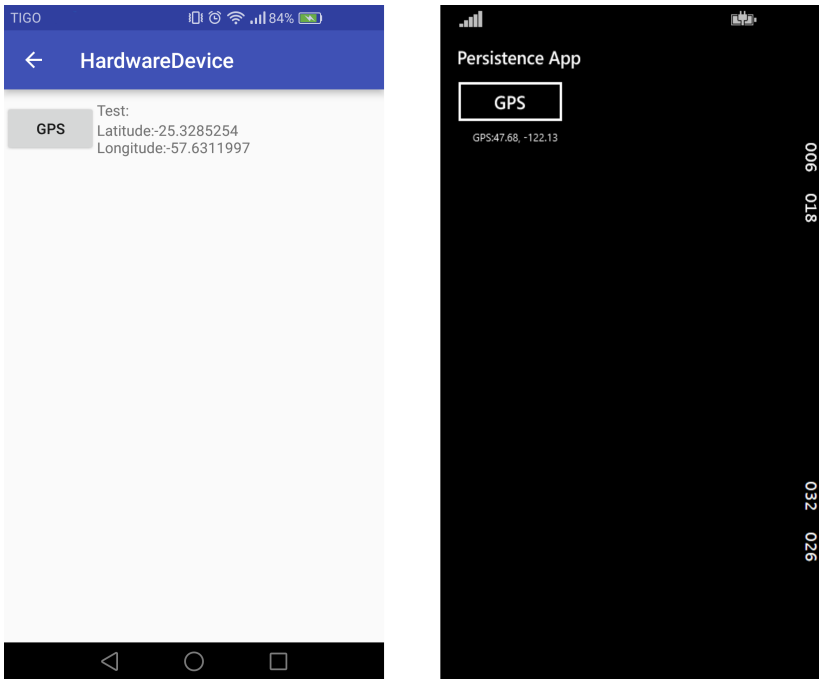


Figura B.8 Sensores definidos. Aplicación *e-market*. *A la izquierda Android, a la derecha Windows Phone*

B.2 Código generado

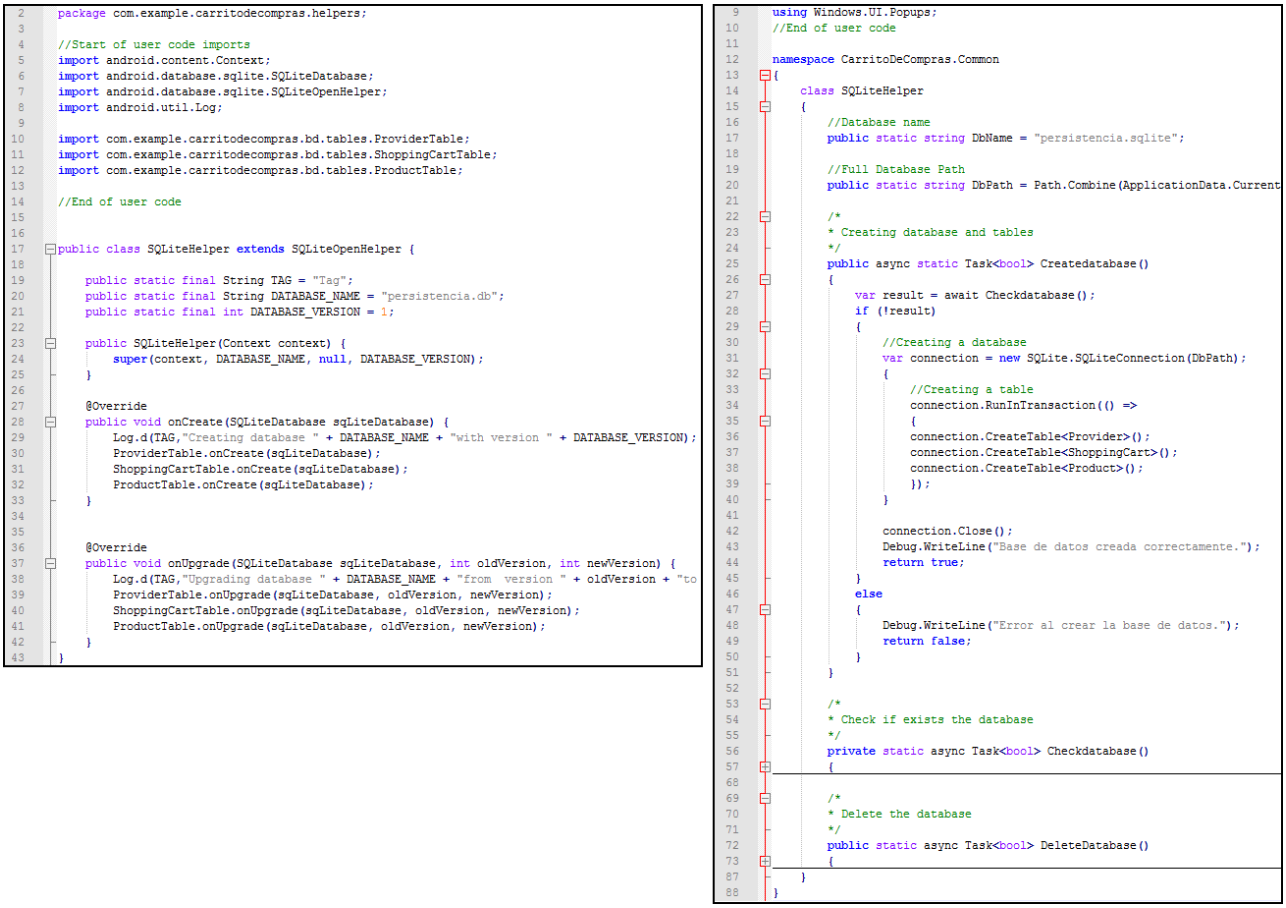
```
[template public generateElement(model: Model)]
[comment @main/]
[let aPackages: Sequence(Package) = model.eAllContents(Package) ]

[generateGeneralAndroidClasses(model)/]
[generateGeneralWindowsClasses(model)/]

[for (aPackage : Package | aPackages)]
[let aClasses: Set(Class) = aPackage.ownedElement->filter(Class) ]
[let p : Package = aPackage.ancestors(Package)->first()]
[comment]Recorremos los paquetes existentes en el modelo. Solo nos interesa dos paquetes:
 DataPersistence y DataProvider[/comment]

    [comment]Si existe el paquete DataPersistence[/comment]
    [if (aPackage.hasStereotype('DataPersistence'))]

        [comment]Genera los beans o modelos de la aplicacion[/comment]
        [beansGenAndroid(aPackage, p.name.toLower())/]
        [beansGenWindows(aPackage)/]

        [comment]Solo creamos este archivo Si existe por lo menos una entidad tipo Database[/comment]
        [if (aPackage.isPackageHasThisPropertyStereotype('persistentEntity', 'persistentType', 'Databas
            [generateDBForAndroid(aPackage, p.name.toLower())/]
            [generateDBForWindows(aPackage, p.name.toUpperFirst())/]
        [/if]

        [comment]Solo creamos este archivo Si existe por lo menos una entidad tipo File[/comment]
        [if (aPackage.isPackageHasThisPropertyStereotype('persistentEntity', 'persistentType', 'File'))
            [generateFilesForAndroid(aPackage, p.name.toLower())/]
            [generateFilesForWindows(aPackage, p.name.toUpperFirst())/]
        [/if]

        [comment]Solo creamos este archivo Si existe por lo menos una entidad tipo KeyValue[/comment]
        [if (aPackage.isPackageHasThisPropertyStereotype('persistentEntity', 'persistentType', 'KeyValu
            [generateKVForAndroid(aPackage, p.name.toLower())/]
            [generateKVForWindows(aPackage, p.name.toUpperFirst())/]
        [/if]
    [/if]

    [comment]Si existe el paquete DataProvider[/comment]
    [if (aPackage.hasStereotype('DataProvider'))]
        [for (aClass : Class | aClasses)]
            [comment]Si la clase tiene el paquete WebServiceInterface[/comment]
            [if (aClass.hasStereotype('WebServiceInterface'))]
                [generateRestAndroid(aClass, p.name.toLower())/]
                [generateRestWindows(aClass, p.name.toUpperFirst())/]
            [/if]

            [comment]Si la clase tiene el paquete HardwareDeviceInterface[/comment]
            [if (aClass.hasStereotype('HardwareDeviceInterface'))]
                [generateSensorsAndroid(aClass, p.name.toLower())/]
                [generateSensorsWindows(aClass, p.name.toUpperFirst())/]
            [/if]
        [/for]
    [/if]
[/let]
[/let]
[/for]
[/let]
[/template]
```

Figura B.9 *Template* principal de las reglas de transformación en Acceleo

```java
public class ShoppingCart  implements Serializable{


    /**
    * the idCart attribute.
    */
    private Integer idCart;


    /**
    * the price attribute.
    */
    private String price;


    /**
    * the quantity attribute.
    */
    private Integer quantity;


    /**
    * the idProduct attribute.
    */
    private Integer idProduct;


    /**
    * the syncTime attribute.
    */
    private BigDecimal syncTime;


    /**
    * Empty Constructor.
    */
    public ShoppingCart (){
    }

    /**
    * Constructor.
    */
    public ShoppingCart (Integer idCart, String price, Integer quantity,
                         Integer idProduct, BigDecimal syncTime){
        this.idCart = idCart;
        this.price = price;
        this.quantity = quantity;
        this.idProduct = idProduct;
        this.syncTime = syncTime;
    }

    public Integer getIdCart() {
       return this.idCart;
     }

    public String getPrice() {
       return this.price;
     }

    public Integer getQuantity() {
       return this.quantity;
     }

    public Integer getIdProduct() {
       return this.idProduct;
     }

    public BigDecimal getSyncTime() {
       return this.syncTime;
     }

     public void setIdCart(Integer idCart) {
        this.idCart = idCart;
     }

     public void setPrice(String price) {
        this.price = price;
     }

     public void setQuantity(Integer quantity) {
        this.quantity = quantity;
     }

     public void setIdProduct(Integer idProduct) {
        this.idProduct = idProduct;
     }

     public void setSyncTime(BigDecimal syncTime) {
        this.syncTime = syncTime;
     }

    public String getTotalCarPrice() {
    // Start of user code getTotalCarPrice
    // TODO should be implemented
    // End of user code
        return "";
    }

    public int getCartProductCount() {
    // Start of user code getCartProductCount
    // TODO should be implemented
    // End of user code
        return 0;
    }
```

```c#
//Start of user code imports
using System;
using System.Runtime.Serialization;
using System.Collections.Generic;
//End of user code

namespace CarritoDeCompras.Model
{
    [DataContract]
    public class ShoppingCart
    {
        //The Id property is marked as the Primary Key
        [SQLite.PrimaryKey, SQLite.AutoIncrement]
        [DataMember]
        public int Id { get; set; }

        [DataMember]
        public int idCart { get; set; }

        [DataMember]
        public string price { get; set; }

        [DataMember]
        public int quantity { get; set; }

        [DataMember]
        public int idProduct { get; set; }

        [DataMember]
        public double syncTime { get; set; }

        /**
        * Empty Constructor.
        */
        public ShoppingCart (){
        }

        /**
        * Constructor.
        */
        public ShoppingCart (int idCart, string price, int quantity,
                             int idProduct, double syncTime){
            this.idCart = idCart;
            this.price = price;
            this.quantity = quantity;
            this.idProduct = idProduct;
            this.syncTime = syncTime;
        }

        public String getTotalCarPrice() {
        // Start of user code getTotalCarPrice
        // TODO should be implemented
        // End of user code
            return "";
        }

        public int getCartProductCount() {
        // Start of user code getCartProductCount
        // TODO should be implemented
        // End of user code
            return 0;
        }
    }
}
```

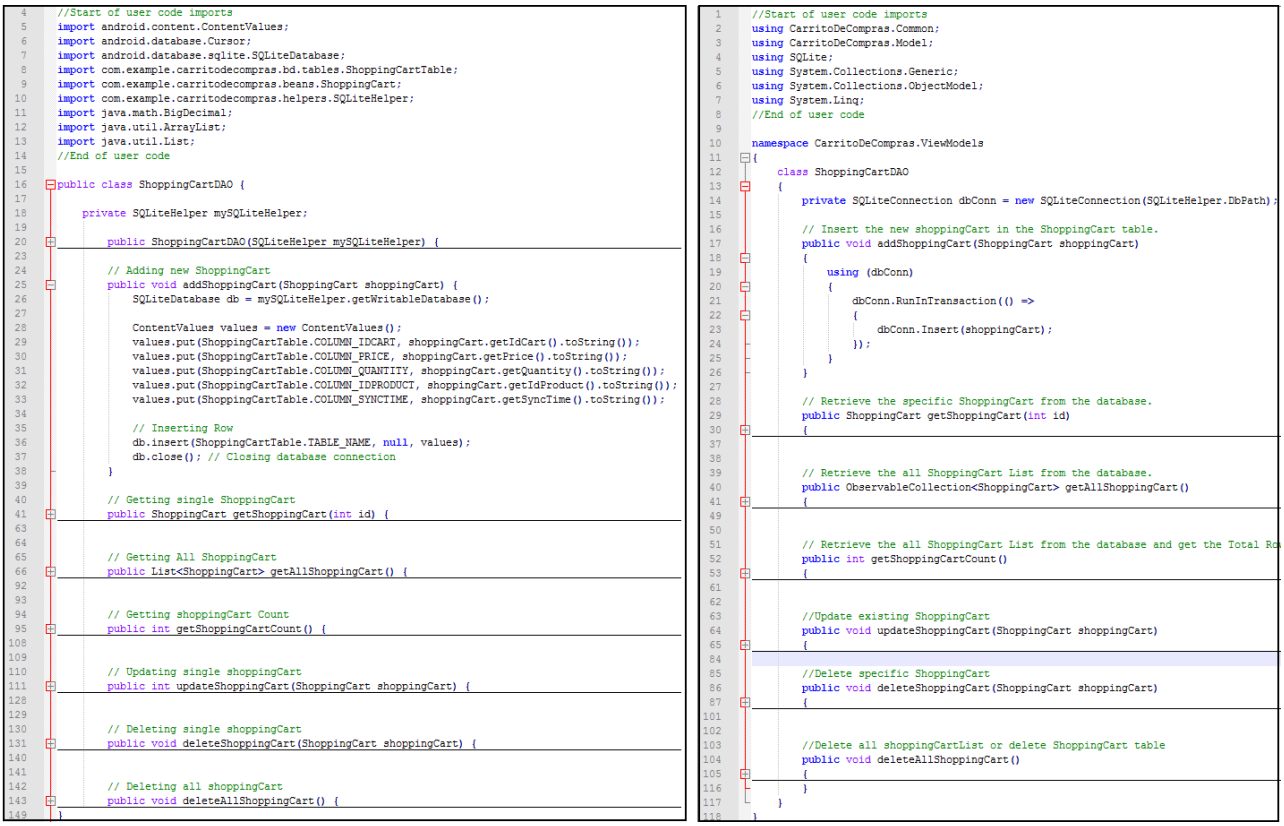
Figura B.10 Modelo de objeto para la entidad persistente *ShoppingCart*. Aplicación *e-market*. *A la izquierda Android (java), a la derecha Windows Phone (C#)*



```java
package com.example.carritodecompras.helpers;

//Start of user code imports
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

import com.example.carritodecompras.bd.tables.ProviderTable;
import com.example.carritodecompras.bd.tables.ShoppingCartTable;
import com.example.carritodecompras.bd.tables.ProductTable;

//End of user code


public class SQLiteHelper extends SQLiteOpenHelper {

    public static final String TAG = "Tag";
    public static final String DATABASE_NAME = "persistencia.db";
    public static final int DATABASE_VERSION = 1;

    public SQLiteHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase sqLiteDatabase) {
        Log.d(TAG,"Creating database " + DATABASE_NAME + "with version " + DATABASE_VERSION);
        ProviderTable.onCreate(sqLiteDatabase);
        ShoppingCartTable.onCreate(sqLiteDatabase);
        ProductTable.onCreate(sqLiteDatabase);
    }


    @Override
    public void onUpgrade(SQLiteDatabase sqLiteDatabase, int oldVersion, int newVersion) {
        Log.d(TAG,"Upgrading database " + DATABASE_NAME + "from  version " + oldVersion + "to
        ProviderTable.onUpgrade(sqLiteDatabase, oldVersion, newVersion);
        ShoppingCartTable.onUpgrade(sqLiteDatabase, oldVersion, newVersion);
        ProductTable.onUpgrade(sqLiteDatabase, oldVersion, newVersion);
    }
}
```

```csharp
using Windows.UI.Popups;
//End of user code

namespace CarritoDeCompras.Common
{
    class SQLiteHelper
    {
        //Database name
        public static string DbName = "persistencia.sqlite";

        //Full Database Path
        public static string DbPath = Path.Combine(ApplicationData.Current

        /*
        * Creating database and tables
        */
        public async static Task<bool> Createdatabase()
        {
            var result = await Checkdatabase();
            if (!result)
            {
                //Creating a database
                var connection = new SQLite.SQLiteConnection(DbPath);
                {
                    //Creating a table
                    connection.RunInTransaction(() =>
                    {
                    connection.CreateTable<Provider>();
                    connection.CreateTable<ShoppingCart>();
                    connection.CreateTable<Product>();
                    });
                }

                connection.Close();
                Debug.WriteLine("Base de datos creada correctamente.");
                return true;
            }
            else
            {
                Debug.WriteLine("Error al crear la base de datos.");
                return false;
            }
        }

        /*
        * Check if exists the database
        */
        private static async Task<bool> Checkdatabase()
        {

        /*
        * Delete the database
        */
        public static async Task<bool> DeleteDatabase()
        {
    }
}
```

Figura B.11 Clase útil *SqliteHelper*. Funcionalidades para el manejo de base de datos. Aplicación *e-market*. *A la izquierda Android (java), a la derecha Windows Phone (C#)*



```
//Start of user code imports
import android.content.ContentValues;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import com.example.carritodecompras.bd.tables.ShoppingCartTable;
import com.example.carritodecompras.beans.ShoppingCart;
import com.example.carritodecompras.helpers.SQLiteHelper;
import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.List;
//End of user code

public class ShoppingCartDAO {

    private SQLiteHelper mySQLiteHelper;

        public ShoppingCartDAO(SQLiteHelper mySQLiteHelper) {

        // Adding new ShoppingCart
        public void addShoppingCart(ShoppingCart shoppingCart) {
            SQLiteDatabase db = mySQLiteHelper.getWritableDatabase();

            ContentValues values = new ContentValues();
            values.put(ShoppingCartTable.COLUMN_IDCART, shoppingCart.getIdCart().toString());
            values.put(ShoppingCartTable.COLUMN_PRICE, shoppingCart.getPrice().toString());
            values.put(ShoppingCartTable.COLUMN_QUANTITY, shoppingCart.getQuantity().toString());
            values.put(ShoppingCartTable.COLUMN_IDPRODUCT, shoppingCart.getIdProduct().toString());
            values.put(ShoppingCartTable.COLUMN_SYNCTIME, shoppingCart.getSyncTime().toString());

            // Inserting Row
            db.insert(ShoppingCartTable.TABLE_NAME, null, values);
            db.close(); // Closing database connection
        }

        // Getting single ShoppingCart
        public ShoppingCart getShoppingCart(int id) {


        // Getting All ShoppingCart
        public List<ShoppingCart> getAllShoppingCart() {


        // Getting shoppingCart Count
        public int getShoppingCartCount() {


        // Updating single shoppingCart
        public int updateShoppingCart(ShoppingCart shoppingCart) {


        // Deleting single shoppingCart
        public void deleteShoppingCart(ShoppingCart shoppingCart) {


        // Deleting all shoppingCart
        public void deleteAllShoppingCart() {
}
```

```
//Start of user code imports
using CarritoDeCompras.Common;
using CarritoDeCompras.Model;
using SQLite;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
//End of user code

namespace CarritoDeCompras.ViewModels
{
    class ShoppingCartDAO
    {
        private SQLiteConnection dbConn = new SQLiteConnection(SQLiteHelper.DbPath);

        // Insert the new shoppingCart in the ShoppingCart table.
        public void addShoppingCart(ShoppingCart shoppingCart)
        {
            using (dbConn)
            {
                dbConn.RunInTransaction(() =>
                {
                    dbConn.Insert(shoppingCart);
                });
            }
        }

        // Retrieve the specific ShoppingCart from the database.
        public ShoppingCart getShoppingCart(int id)


        // Retrieve the all ShoppingCart List from the database.
        public ObservableCollection<ShoppingCart> getAllShoppingCart()


        // Retrieve the all ShoppingCart List from the database and get the Total Ro
        public int getShoppingCartCount()


        //Update existing ShoppingCart
        public void updateShoppingCart(ShoppingCart shoppingCart)


        //Delete specific ShoppingCart
        public void deleteShoppingCart(ShoppingCart shoppingCart)


        //Delete all shoppingCartList or delete ShoppingCart table
        public void deleteAllShoppingCart()
        }
    }
}
```

Figura B.12 DAO de la tabla *ShoppingCart*. Aplicación *e-market*. *A la izquierda Android (java), a la derecha Windows Phone (C#)*

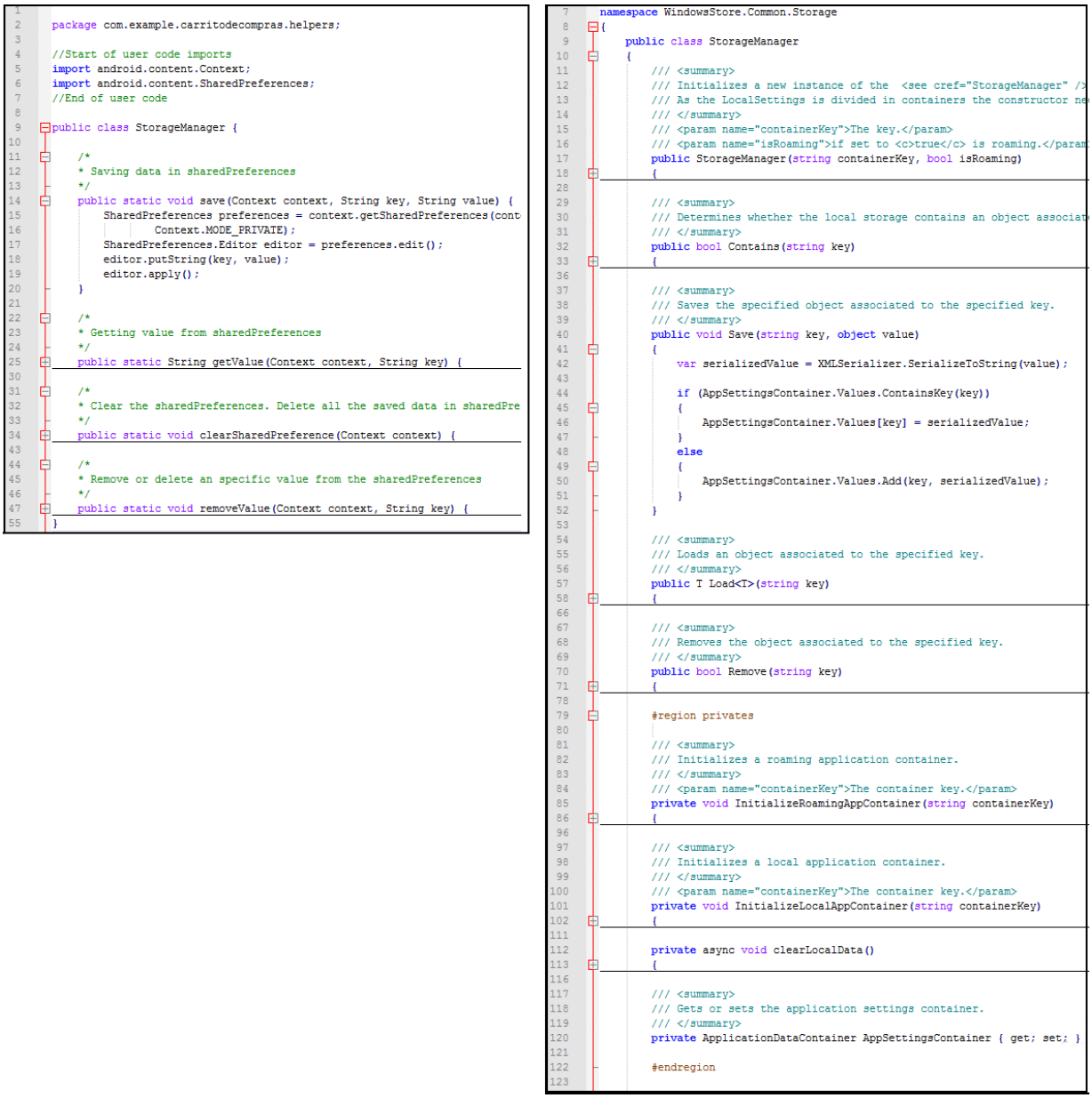
```java
package com.example.carritodecompras.bd.tables;


//Start of user code imports

import android.content.ContentResolver;
import android.database.sqlite.SQLiteDatabase;
//End of user code

public class ShoppingCartTable {
    private ContentResolver contentResolver;

    //    DATABASE TABLE
    public static final String TABLE_NAME = "shoppingCart";

    public static final String COLUMN_ID = "_id";
    public static final String COLUMN_IDCART = "idCart";
    public static final String COLUMN_PRICE = "price";
    public static final String COLUMN_QUANTITY = "quantity";
    public static final String COLUMN_IDPRODUCT = "idProduct";
    public static final String COLUMN_SYNCTIME = "syncTime";

    //    DATABASE CREATION SQL STATEMENT
    private static final String CREATE_TABLE = "create table "
            + TABLE_NAME
            + "("
            + COLUMN_ID + " integer primary key autoincrement,"
            + COLUMN_IDCART + " integer NOT NULL ,"
            + COLUMN_PRICE + " text NOT NULL ,"
            + COLUMN_QUANTITY + " integer NOT NULL ,"
            + COLUMN_IDPRODUCT + " integer NOT NULL ,"
            + COLUMN_SYNCTIME + " numeric);";


    public ShoppingCartTable(ContentResolver contentResolver) {
        this.contentResolver = contentResolver;
    }

    public static void onCreate(SQLiteDatabase database) {
        database.execSQL(CREATE_TABLE);
    }

    public static void onUpgrade(SQLiteDatabase database, int oldVersion, int newVersion) {
        database.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
        onCreate(database);
    }

}
```

Figura B.13 Definición de la tabla *ShoppingCart* para Android. Aplicación *e-market*

```java
import java.util.List;
//End of user code

public class FilesHelper {

    public final static String FILE_EXTENSION_SEPARATOR = ".";

    private FilesHelper() {
        throw new AssertionError();
    }

    /**
     * read file
     */
    public static StringBuilder readFile(String filePath, String charsetName) {

    /**
     * write file
     */
    public static boolean writeFile(String filePath, String content, boolean append) {

    /**
     * write file
     */
    public static boolean writeFile(String filePath, List<String> contentList, boolean append) {

    /**
     * write file, the string will be written to the begin of the file
     */
    public static boolean writeFile(String filePath, String content) {

    /**
     * write file, the string list will be written to the begin of the file
     *
     * @param filePath
     * @param contentList
     * @return
     */
    public static boolean writeFile(String filePath, List<String> contentList) {

    /**
     * write file, the bytes will be written to the begin of the file
     */
    public static boolean writeFile(String filePath, InputStream stream) {

    /**
     * write file
     */
    public static boolean writeFile(String filePath, InputStream stream, boolean append) {

    /**
     * write file, the bytes will be written to the begin of the file
     */
    public static boolean writeFile(File file, InputStream stream) {

    /**
     * write file
     */
    public static boolean writeFile(File file, InputStream stream, boolean append) {

    /**
     * move file
     */
    public static void moveFile(String sourceFilePath, String destFilePath) {

    /**
     * move file
     */
    public static void moveFile(File srcFile, File destFile) {

    /**
     * copy file
     */
    public static boolean copyFile(String sourceFilePath, String destFilePath) {
```

```csharp
//End of user code

namespace CarritoDeCompras.Common
{

    class FilesHelper
    {
        private static StringBuilder folderContents;

        private const string FOLDER_PREFIX = "\\";
        private const int PADDING_FACTOR = 3;
        private const char SPACE = ' ';

        /*
        * Creating new folders and files
        */
        public static async Task createFolder(string folderName)
        {

        /*
        * create a new file in the local folder
        */
        public static async Task createFile(string newFileName)
        {

        /*
        * Save a photo to the app's local folder
        */
        public static async Task<string> SavePhoto(Stream photoToSave, string fileName)
        {

        /*
        * Write a text file to the app's local folder
        */
        public static async Task<string> WriteTextFile(string filename, string contents)
        {
            StorageFolder localFolder = ApplicationData.Current.LocalFolder;
            StorageFile textFile = await localFolder.CreateFileAsync(filename,
                                                    CreationCollisionOption.ReplaceExisti
            using (IRandomAccessStream textStream = await textFile.OpenAsync(FileAccessMode.ReadW
            {
                using (DataWriter textWriter = new DataWriter(textStream))
                {
                    textWriter.WriteString(contents);
                    await textWriter.StoreAsync();
                }
            }
            return textFile.Path;

        }

        /*
        * Read the contents of a text file from the app's local folder
        */
        public static async Task<string> ReadTextFile(string filename)
        {


        /*
        * Begin recursive enumeration of files and folders
        */
        public static async Task<string> EnumerateFilesAndFolders(StorageFolder rootFolder)
        {

        /*
        * Continue recursive enumeration of files and folders
        */
        private static async Task ListFilesInFolder(StorageFolder folder, int indentationLevel)
        {

    }
}
```

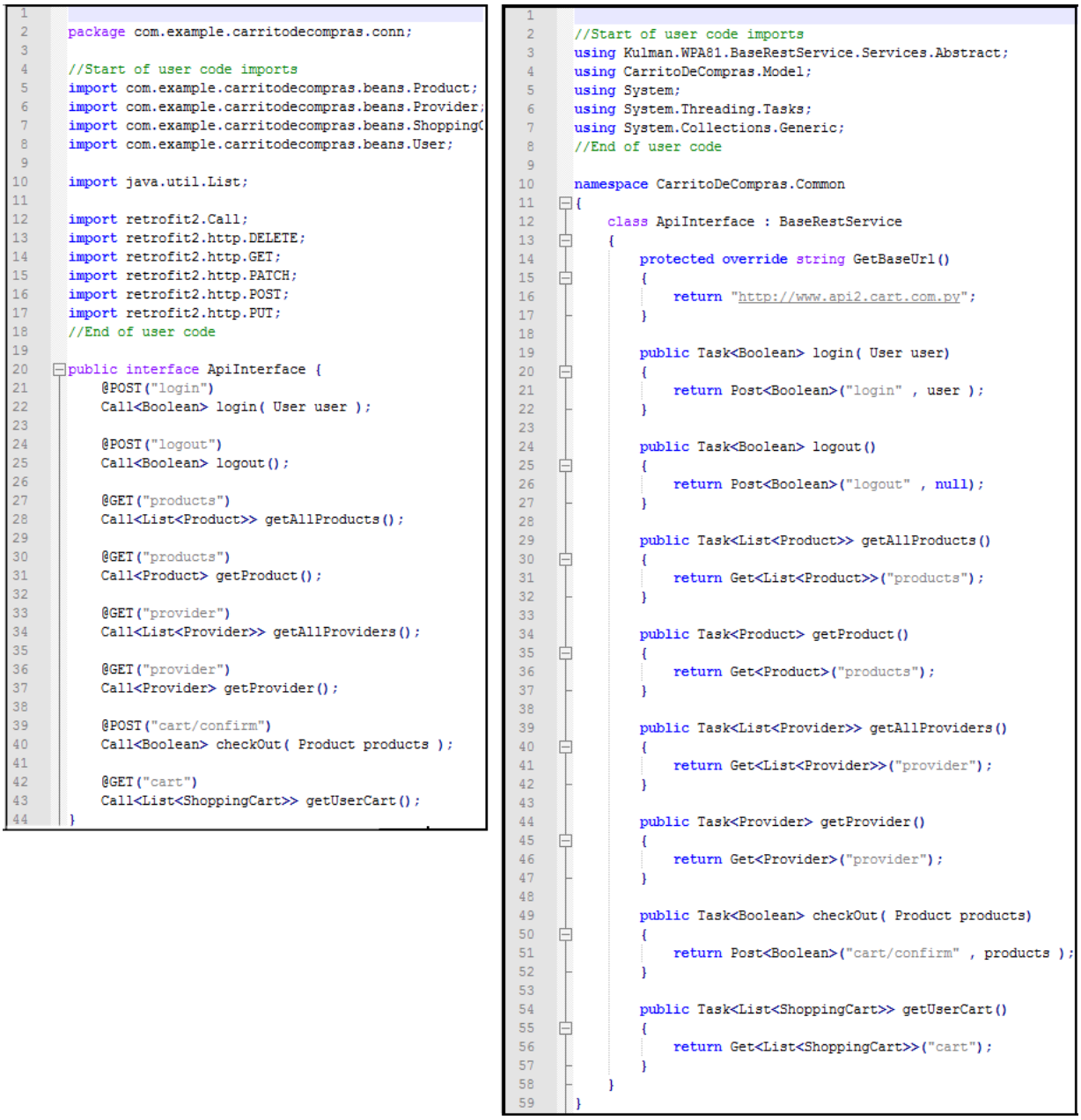
Figura B.14 Clase útil *FileHelper*. Funcionalidades para el manejo de archivos. Aplicación *e-market*. *A la izquierda Android (java), a la derecha Windows Phone (C#)*



```java
package com.example.carritodecompras.helpers;

//Start of user code imports
import android.content.Context;
import android.content.SharedPreferences;
//End of user code

public class StorageManager {

    /*
    * Saving data in sharedPreferences
    */
    public static void save(Context context, String key, String value) {
        SharedPreferences preferences = context.getSharedPreferences(cont
                Context.MODE_PRIVATE);
        SharedPreferences.Editor editor = preferences.edit();
        editor.putString(key, value);
        editor.apply();
    }

    /*
    * Getting value from sharedPreferences
    */
    public static String getValue(Context context, String key) {

    /*
    * Clear the sharedPreferences. Delete all the saved data in sharedPre
    */
    public static void clearSharedPreference(Context context) {

    /*
    * Remove or delete an specific value from the sharedPreferences
    */
    public static void removeValue(Context context, String key) {
}
```

```csharp
namespace WindowsStore.Common.Storage
{
    public class StorageManager
    {
        /// <summary>
        /// Initializes a new instance of the  <see cref="StorageManager" />
        /// As the LocalSettings is divided in containers the constructor ne
        /// </summary>
        /// <param name="containerKey">The key.</param>
        /// <param name="isRoaming">if set to <c>true</c> is roaming.</param
        public StorageManager(string containerKey, bool isRoaming)
        {

        /// <summary>
        /// Determines whether the local storage contains an object associat
        /// </summary>
        public bool Contains(string key)
        {

        /// <summary>
        /// Saves the specified object associated to the specified key.
        /// </summary>
        public void Save(string key, object value)
        {
            var serializedValue = XMLSerializer.SerializeToString(value);

            if (AppSettingsContainer.Values.ContainsKey(key))
            {
                AppSettingsContainer.Values[key] = serializedValue;
            }
            else
            {
                AppSettingsContainer.Values.Add(key, serializedValue);
            }
        }

        /// <summary>
        /// Loads an object associated to the specified key.
        /// </summary>
        public T Load<T>(string key)
        {

        /// <summary>
        /// Removes the object associated to the specified key.
        /// </summary>
        public bool Remove(string key)
        {

        #region privates

        /// <summary>
        /// Initializes a roaming application container.
        /// </summary>
        /// <param name="containerKey">The container key.</param>
        private void InitializeRoamingAppContainer(string containerKey)
        {

        /// <summary>
        /// Initializes a local application container.
        /// </summary>
        /// <param name="containerKey">The container key.</param>
        private void InitializeLocalAppContainer(string containerKey)
        {

        private async void clearLocalData()
        {

        /// <summary>
        /// Gets or sets the application settings container.
        /// </summary>
        private ApplicationDataContainer AppSettingsContainer { get; set; }

        #endregion
```

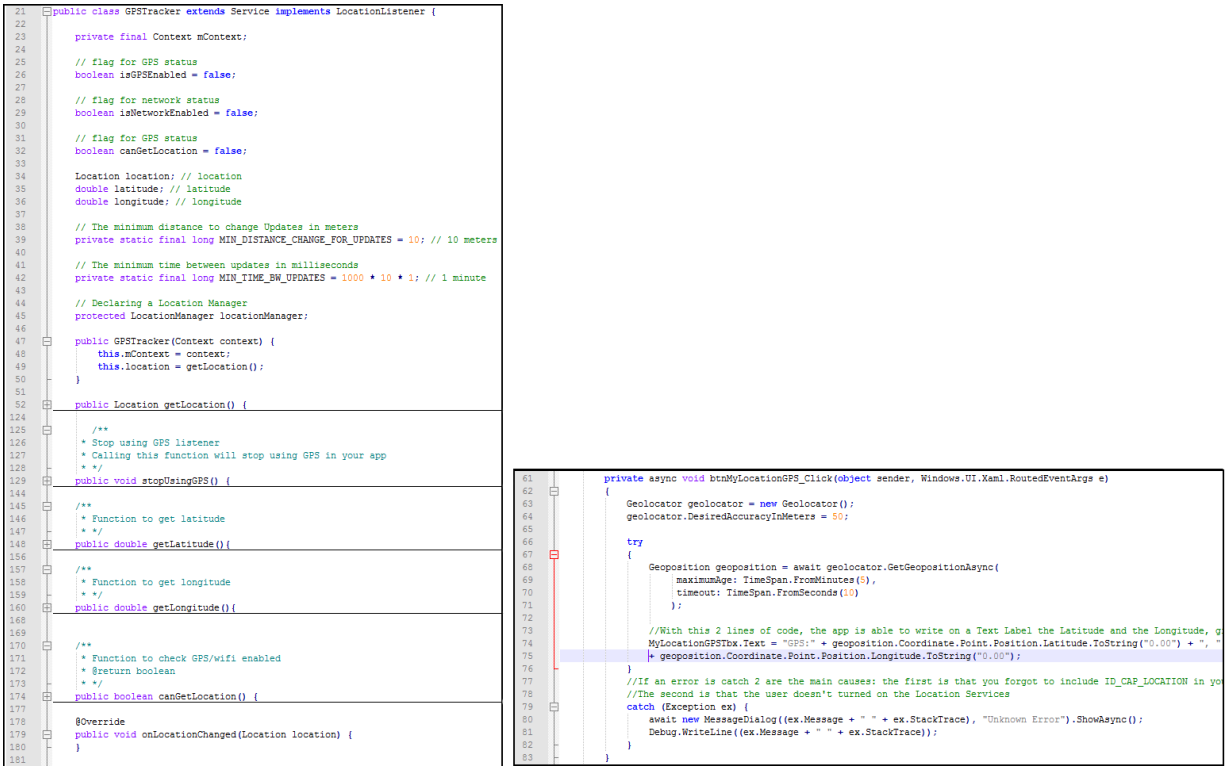
Figura B.15 Clase útil *StorageManager*. Funcionalidades para el manejo de pares clave-valor. Aplicación *e-market*. *A la izquierda Android (java), a la derecha Windows Phone (C#)*



```java
package com.example.carritodecompras.conn;

//Start of user code imports
import com.example.carritodecompras.beans.Product;
import com.example.carritodecompras.beans.Provider;
import com.example.carritodecompras.beans.ShoppingC
import com.example.carritodecompras.beans.User;

import java.util.List;

import retrofit2.Call;
import retrofit2.http.DELETE;
import retrofit2.http.GET;
import retrofit2.http.PATCH;
import retrofit2.http.POST;
import retrofit2.http.PUT;
//End of user code

public interface ApiInterface {
    @POST("login")
    Call<Boolean> login( User user );

    @POST("logout")
    Call<Boolean> logout();

    @GET("products")
    Call<List<Product>> getAllProducts();

    @GET("products")
    Call<Product> getProduct();

    @GET("provider")
    Call<List<Provider>> getAllProviders();

    @GET("provider")
    Call<Provider> getProvider();

    @POST("cart/confirm")
    Call<Boolean> checkOut( Product products );

    @GET("cart")
    Call<List<ShoppingCart>> getUserCart();
}
```

```csharp
//Start of user code imports
using Kulman.WPA81.BaseRestService.Services.Abstract;
using CarritoDeCompras.Model;
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
//End of user code

namespace CarritoDeCompras.Common
{
    class ApiInterface : BaseRestService
    {
        protected override string GetBaseUrl()
        {
            return "http://www.api2.cart.com.py";
        }

        public Task<Boolean> login( User user)
        {
            return Post<Boolean>("login" , user );
        }

        public Task<Boolean> logout()
        {
            return Post<Boolean>("logout" , null);
        }

        public Task<List<Product>> getAllProducts()
        {
            return Get<List<Product>>("products");
        }

        public Task<Product> getProduct()
        {
            return Get<Product>("products");
        }

        public Task<List<Provider>> getAllProviders()
        {
            return Get<List<Provider>>("provider");
        }

        public Task<Provider> getProvider()
        {
            return Get<Provider>("provider");
        }

        public Task<Boolean> checkOut( Product products)
        {
            return Post<Boolean>("cart/confirm" , products );
        }

        public Task<List<ShoppingCart>> getUserCart()
        {
            return Get<List<ShoppingCart>>("cart");
        }
    }
}
```

Figura B.16 Interfaz REST. Aplicación *e-market*. *A la izquierda Android (java), a la derecha Windows Phone (C#)*



```java
public class GPSTracker extends Service implements LocationListener {

    private final Context mContext;

    // flag for GPS status
    boolean isGPSEnabled = false;

    // flag for network status
    boolean isNetworkEnabled = false;

    // flag for GPS status
    boolean canGetLocation = false;

    Location location; // location
    double latitude; // latitude
    double longitude; // longitude

    // The minimum distance to change Updates in meters
    private static final long MIN_DISTANCE_CHANGE_FOR_UPDATES = 10; // 10 meters

    // The minimum time between updates in milliseconds
    private static final long MIN_TIME_BW_UPDATES = 1000 * 10 * 1; // 1 minute

    // Declaring a Location Manager
    protected LocationManager locationManager;

    public GPSTracker(Context context) {
        this.mContext = context;
        this.location = getLocation();
    }

    public Location getLocation() {

        /**
     * Stop using GPS listener
     * Calling this function will stop using GPS in your app
     * */
    public void stopUsingGPS() {

    /**
     * Function to get latitude
     * */
    public double getLatitude(){

    /**
     * Function to get longitude
     * */
    public double getLongitude(){


    /**
     * Function to check GPS/wifi enabled
     * @return boolean
     * */
    public boolean canGetLocation() {

    @Override
    public void onLocationChanged(Location location) {
    }
```

```java
private async void btnMyLocationGPS_Click(object sender, Windows.UI.Xaml.RoutedEventArgs e)
{
    Geolocator geolocator = new Geolocator();
    geolocator.DesiredAccuracyInMeters = 50;

    try
    {
        Geoposition geoposition = await geolocator.GetGeopositionAsync(
            maximumAge: TimeSpan.FromMinutes(5),
            timeout: TimeSpan.FromSeconds(10)
            );

        //With this 2 lines of code, the app is able to write on a Text Label the Latitude and the Longitude, g
        MyLocationGPSTbx.Text = "GPS:" + geoposition.Coordinate.Point.Position.Latitude.ToString("0.00") + ", "
        + geoposition.Coordinate.Point.Position.Longitude.ToString("0.00");
    }
    //If an error is catch 2 are the main causes: the first is that you forgot to include ID_CAP_LOCATION in yo
    //The second is that the user doesn't turned on the Location Services
    catch (Exception ex) {
        await new MessageDialog((ex.Message + " " + ex.StackTrace), "Unknown Error").ShowAsync();
        Debug.WriteLine((ex.Message + " " + ex.StackTrace));
    }
}
```

Figura B.17 Manejo del GPS. Aplicación *e-market*. *A la izquierda Android (java), a la derecha Windows Phone (C#)*

```java
        //Handle incoming data
        Intent intent = getIntent();
        String action = intent.getAction();
        String type = intent.getType();

        //all the data type options to receive
        if (Intent.ACTION_SEND.equals(action) && type != null) {

            if ("text/plain".equals(type)) {
                handleSendText(intent, "text"); // Handle text being sent
            }

        }
```

```java
//End of user code

namespace CarritoDeCompras.Views
{
    public sealed partial class ShareTargetPageView : Page
    {
        ShareOperation operation = null;

        public ShareTargetPageView()
        {
            this.InitializeComponent();
        }

        protected override async void OnNavigatedTo(NavigationEventArgs e)
        {
            operation = (ShareOperation)e.Parameter;

                if (operation.Data.Contains(StandardDataFormats.WebLink)) //URI
                {
                    var uri = await operation.Data.GetWebLinkAsync();
                    if (uri != null)
                    {
                        MessageDialog ms = new MessageDialog("WebLink: " + uri.A
                        await ms.ShowAsync();
                    }
                }
        }

        private void Button_Click(object sender, RoutedEventArgs e)
        {
            operation.ReportCompleted();
        }
    }
}
```

Figura B.18 Interacción con otras aplicaciones. Aplicación *e-market*. *A la izquierda Android (java), a la derecha Windows Phone (C#)*